THE UNIVERSITY OF MICHIGAN

Technical Report 15

CONCOMP

December 1968

AN EXECUTIVE SYSTEM FOR A DEC 339 COMPUTER DISPLAY TERMINAL

James H. Jackson

196

T H E U N I V E R S I T Y O F M I C H I G A N

Technical Report 15

AN EXECUTIVE SYSTEM FOR A DEC 339 COMPUTER DISPLAY TERMINAL

J.mes H. Jackson

CONCOMP: Research in Conversational Use of Computers
F. H. Westervelt, Project Director
ORA Project 07449

Supported by:
ADVANCED RESEARCH PROJECTS AGENCY
DEPARTMENT OF DEFENSE
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

Administered through:
OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

December 1968

ABSTRACT

This report describes a real-time multiprogramming software system for a DEC 339 computer display terminal, which may communicate with an external computer through a serial-synchronous data set. The system is designed to support both programs which require the attention of an external computer while they are being executed and programs which are independent of external computation service. For either type of program, the entire graphics support is provided by the 339 system, but the interpretation of the relations implied by the graphics may be performed either in the 339 or in an external computer. Multiprogramming facility is provided to facilitate effective use of I/O devices in order to cope with the demands of a real-time environment.

TABLE OF CONTENTS

BLANK PAGE

1. INTRODUCTION

The objective of this report is to describe the conceptual organization of the SEL (Systems Engineering Laboratory's) Executive System for a 339 computer display terminal, as well as to provide a manual for its use. More specifically, the hardware configuration for which the System was designed consists of the following items (plus necessary interfaces, multiplexors, etc.):

DEC PDP-9 with at least two 8192-word memory banks
DEC KE09A extended arithmetic element
DEC 338 display control (less PDP-8)
DEC AF01B A/D converter
DEC AA01A D/A converter
AT&T 201A data set

The System provides both a multiprogramming capability (based on I/O slicing, rather than time-slicing) and a complete set of operators for maintaining a highly structured display file and for interrogating it for relational properties.

Since an on-line operator tends to produce a burst of inputs and then to be idle for a relatively long period of time, appropriate feedback to each input must be provided rapidly if the operator is to be allowed to proceed at his own rate. If the terminal were not multiprogrammed, the processing of one input would have to be completed before processing of the next could be begun. Consequently, bursts of operator activity could not be effectively handled by this scheme. However, if a multiprogramming system (where the users of the system are programs which respond to various inputs) were used, feedback to each input could be produced almost immediately, and the remaining (and usually time-consuming) part of the processing could be deferred until a later time.

Bandwidth limitations on the data link between the remote computer and the central timesharing system suggest that programs be distributed between the central computer and the remote computer such that dataphone traffic is minimized (subject to the constraint of the capacity of the remote machine). In terms of a remote display terminal, this usually means that the relations implied by a display file, rather than the display file itself, be transmitted. For this reason, the remote system should provide a facility for constructing a display file based partly on relational information, and for interrogating a display file for relational information.

A general discussion of the organization of the System and detailed discussions of the various system subroutines and the idle-time task follow. A complete listing of the System is given in Appendix A, a summary of system subroutines is given in Appendix B, a summary of all IOT instructions pertinent to the hardware configuration is given in Appendix C, and a brief description of the assembly language used in the examples is given in Appendix D.

2. SYSTEM ORGANIZATION

2.1 Bootstrap Arrangement

The System should be loaded by the following procedure:

1) Place the system tape in the reader.
2) Set all switches to 0 (down).
3) Depress the read-in key.

This procedure causes the first record, which is written in hardware RIM format, to be read, and the computer to be started at the last location loaded. The record read is the bootstrap loader represented by the following assembly code:

```
$ORG      0
IOT       144      SELECT READER IN BINARY MODE
IOT       101      SKIP ON READER FLAG
JMP       *-1      WAIT FOR READER FLAG
IOT       112      READ READER BUFFER
DAC*      10       LOAD A WORD
JMP       0        READ NEXT WORD
HLT
HLT
$DC       17731    INITIAL INDEX VALUE
JMP       0        START BOOTSTRAP LOADER
```

The bootstrap loader is capable of loading one binary block (Section 3.4.2) starting at location 17732_8, but is not capable of detecting the end of the block. However, the block which immediately follows the bootstrap loader on the system tape is loaded into locations $17732_8,\ldots,17777_8$, 0. The word loaded into location 0 is a JMP instruction to the beginning of a more sophisticated loader, which is contained in the block read by the bootstrap loader.

The loader loaded by the bootstrap loader is capable of loading an arbitrary number of binary blocks, and it is this

loader which loads the System. Immediately following the last block of the System is a one-word block which modifies the loader and causes execution of the System to begin.

At the end of the loading process, the System occupies locations $0\text{-}11777_8$, and the bootstrap loader and system loader are no longer usable. (The storage occupied by the system loader is salvaged by the System for display structure use at a later time.)

2.2 Tasks

Each program written to run with the System is called a "task" and is identified by its entry point. The System maintains a task queue, each entry of which consists of the entry point for the task, together with other information required to determine the eligibility of the task or to restore the contents of certain registers before the task is executed. Whenever execution of a task is begun, the task is removed from the task queue.

A task is entered by a JMP instruction (rather than a JMS instruction, as in some other similar systems) and is subject to the following restrictions:

1) No user task may contain an IOT instruction.

2) No user task may store in core bank 0. (No user task should be loaded into core bank 0. Locations $12000_8\text{-}17777_8$ are used by the System to store the display structure.)

3) A task which uses an allocatable I/O device (via system subroutines) must allocate the device before calling the system subroutine to use it, and must release the device before terminating. (The task may allocate and/or release the device implicitly by insuring that another task is scheduled to perform the function.)

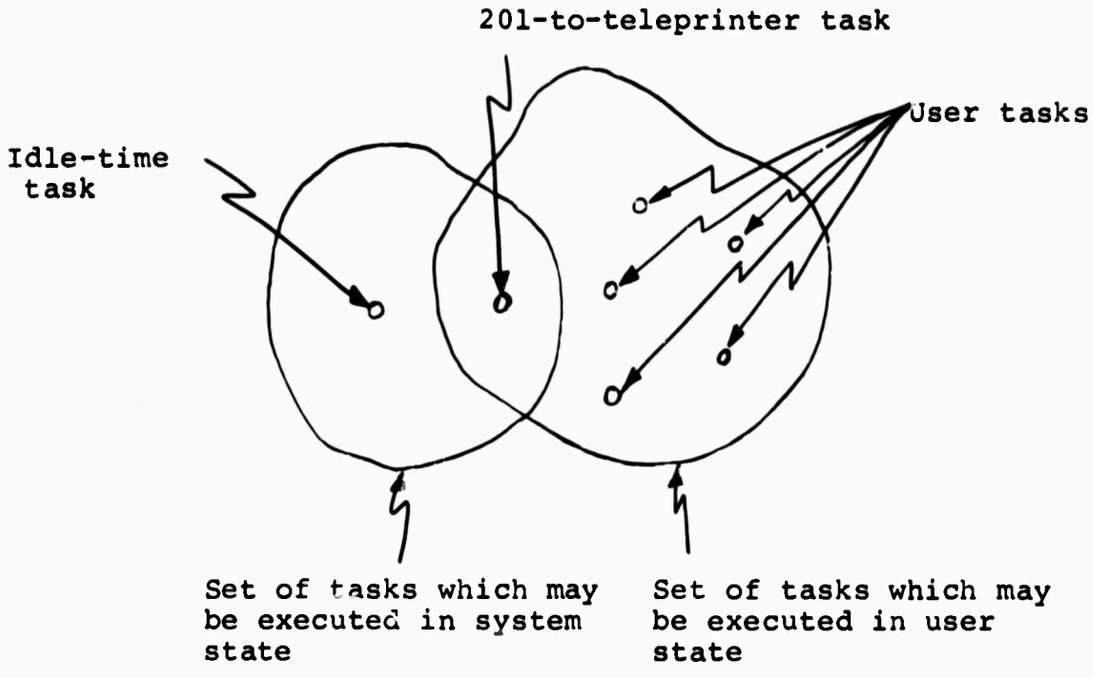
2.3 States of the System

At any instant, the System is operating in one of two states:

1) System state--A special system task, called the idle-time task (Section 4), is executed. However, an incoming message from the 201A dataphone which is not directed to a user task will cause the 201-to-teleprinter task (Section 3.4.1) to be scheduled.

2) User state--All scheduled user tasks are executed and the idle-time task is not executed. The 201-to-teleprinter task is scheduled when necessary as in system state.

The states of the System may be depicted by the following diagram:



2.4 Entering System State

Whenever one of the following events occurs, the System is reinitialized (i.e., all I/O activity is stopped, the task queue and all buffers are cleared, and all I/O devices are

released), and system state is entered:

1) The System is reloaded.

2) The currently executing user task terminates with the task queue empty, and all output buffers become empty.

3) An unidentifiable interrupt occurs.

4) The manual interrupt button is pressed. (The manual interrupt is used by the operator to reinitialize the System in case of emergency.)

5) The task queue overflows.

6) The program is started at location 22_8 via the panel switches.

7) An illegal instruction (operation code 00_8) is executed.

Immediately after system state is entered, a comment describing which one of the above events occurred is typed on the teletype, and, if enough free display storage remains, it is displayed on the screen. Reinitializing the System does not include clearing the display storage area, but it does cause the active structure to be detached from the highest active level (Section 3.9).

3. SYSTEM SUBROUTINES

Sections 3.1 through 3.11 describe the various system subroutines which are callable from user tasks. The entry point to each subroutine occupies a fixed position in a vector such that the actual code for the subroutine may be relocated (by some future modification of the System) without requiring user tasks to be reassembled. Since the System occupies core bank 0 and user tasks cannot be loaded into bank 0, system subroutines must be called via an indirect reference, i.e., if α is the symbolic name of a system subroutine, a call to α is written in the following form:

```
JMS* =α
```

Most of the system subroutines return immediately after the JMS instructions which call them. (Parameters are passed in the AC and MQ.) However, several subroutines have "failure returns," i.e., a return is made immediately after the location containing the JMS instruction if the function which the subroutine must perform cannot be performed. If the subroutine succeeds, return is made to the next location. The two types of calling sequences may be illustrated as follows:

Subroutine with no failure return:

```
JMS* =α
------      (return)
```

Subroutine with failure return:

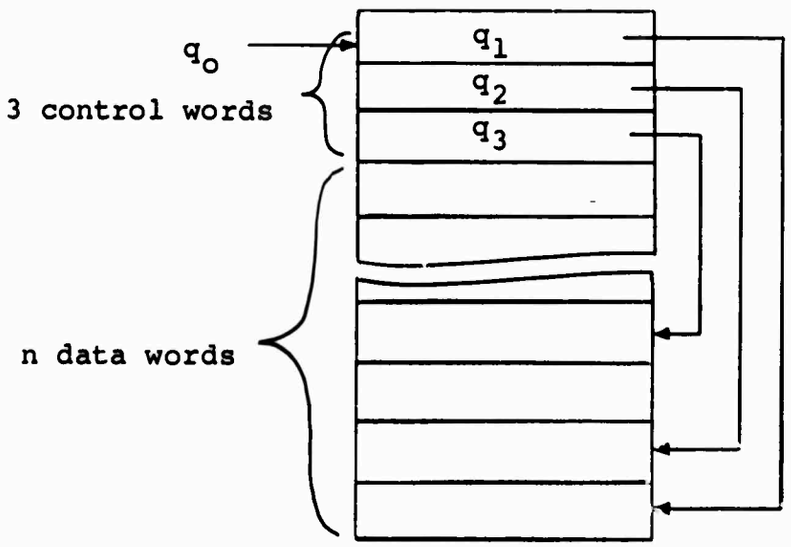
```
JMS* =α
------      (failure return)
------      (success return)
```

A subroutine which has a failure return is denoted by an asterisk (*) appended to its symbolic name in Sections 3.1 through 3.11. (The asterisk is not part of the symbolic name.)

3.1 Word Queues

The basic structure which supports cyclic I/O buffering and task scheduling in the System is a word queue. This structure consists of a block of three words, called control words, followed by n data words and has the properties of both a first-in first-out (FIFO) queue and a last-in first-out (LIFO) queue.

A word queue is represented in core as shown by the following diagram:



The symbols in the diagram are interpreted as follows:

q_o = Address of the word queue. By convention, this is the address of the first control word.

q_1 = Pointer to the physically last data word in the queue.

q_2 = Pointer to the last word put into the queue (FIFO sense).

q_3 = Pointer to the last word taken out of the queue.

The word queue is empty whenever $q_2 = q_3$, and it is full whenever $q_3 = q_2 + 1$ or $q_3 = q_o + 3$ and $q_2 = q_1$. The maximum number of words which may be stored in the queue is then $n - 1$.

The cyclic nature of the word queue requires that the terms incrementing and decrementing a pointer be defined for this structure. A pointer q is "incremented" if it is modified so that it takes on the value

$$q' = \begin{cases} q + 1, & \text{if } q \neq q_1 \\ q_o + 3, & \text{if } q = q_1 \end{cases}$$

A pointer q is "decremented" if it is modified so that it takes on the value

$$q'' = \begin{cases} q - 1, & \text{if } q \neq q_o + 3 \\ q_1, & \text{if } q = q_o + 3 \end{cases}$$

The following system subroutines have been defined for managing word queues:

Q.C - The word queue whose address is given in bits 3-17 of the AC is cleared. (q_2 and q_3 are both set equal to q_1 .)

Q.I*- The word given in the MQ is added in LIFO fashion to the word queue whose address is given in bits 3-17 of the AC. (The word to be queued is stored in the location which q_3 references, and q_3 is decremented.) A failure return is made if the queue is full before the operation is attempted.

Q.A*- The word given in the MQ is added in FIFO fashion to the word queue whose address is given in bits 3-17 of the AC. (q_2 is incremented and the word to be queued is stored in the location which the resulting q_2 references.) A failure return is

made if the queue is full before the operation is attempted.

Q.F*- A word is fetched from the word queue whose address is given in bits 3-17 of the AC and is returned in the AC. (q_3 is incremented, and the word stored in the location which the resulting q_3 references is fetched.) A failure return is made if the word queue is empty before the operation is attempted.

A word queue may be constructed by defining only the pointers q_0 and q_1, since, if the queue is cleared (via Q.C) before it is used, the pointers q_2 and q_3 will be automatically established. For example, the word queue whose address is Q may be constructed by the following two statements, where ε is an expression whose value is n + 2:

```
Q    $DC   *+ε
     $DS   ε
```

As an example of the manipulation (but not application) of word queues, consider a task, whose entry point is TASK, which stores sequential integers on a first-in, first-out basis in the word queue FIFO until the queue is full, and then copies words from FIFO into another word queue LIFO on a last-in, first-out basis. Both FIFO and LIFO will be assumed to have a capacity of X words, where X is a predefined symbol. An algorithm for this task is given below. (T.F is described in Section 3.2.)

```
TASK      LAC   =FIFO        GET ADDRESS OF FIFO QUEUE
          JMS*  =Q.C         CLEAR FIFO QUEUE
          LAC   =LIFO        GET ADDRESS OF LIFO QUEUE
          JMS*  =Q.C         CLEAR LIFO QUEUE
          DZM   COUNT        START COUNTING AT ZERO
LOOP1     LAC   COUNT        GET VALUE OF INTEGER
          LMQ                SET UP PARAMETER
          LAC   =FIFO        GET ADDRESS OF FIFO QUEUE
```

```
        JMS*   =Q.A          ADD INTEGER TO QUEUE
        JMP    LOOP2         COPY INTO OTHER QUEUE
        ISZ    COUNT         INCREMENT COUNTER
        JMP    LOOP1         QUEUE NEXT INTEGER
LOOP2   LAC    =FIFO         GET ADDRESS OF FIFO QUEUE
        JMS*   =Q.F          FETCH WORD FROM QUEUE
        JMS*   =T.F          TERMINATE TASK
        LMQ                  SET UP PARAMETER
        LAC    =LIFO         GET ADDRESS OF LIFO QUEUE
        JMS*   =Q.I          INSERT WORD ON QUEUE
        $DC    0             PROGRAM SHOULD NEVER GET HERE
        JMP    LOOP2         COPY NEXT WORD
FIFO    $DC    *+X+3
        $DS    X+3
LIFO    $DC    *+X+3
        $DS    X+3
```

3.2 Task Scheduling and I/O Device Allocation

The following system subroutines have been defined for controlling task scheduling:

- T.S - The task whose address appears in bits 3-17 in the AC is scheduled for execution.
- T.P - The task whose entry point is the location immediately preceding the call to T.P is scheduled for execution, and execution of the task which called T.P is terminated.
- T.F - Execution of the task which called T.F is terminated.

As an example of the use of these system subroutines, consider a task, whose entry point is SCHED, which schedules the two tasks TASK1 and TASK2 after a nonzero value is stored (by some other task) in location SWITCH. One algorithm for this task is the following:

```
SCHED     JMS    CHECK          SKIP IF SWITCH IS SET
          JMS*   =T.P           WAIT FOR SWITCH TO BE SET
          LAC    =TASK1         GET ADDRESS OF FIRST TASK
          JMS*   =T.S           SCHEDULE FIRST TASK
          LAC    =TASK2         GET ADDRESS OF SECOND TASK
          JMS*   =T.S           SCHEDULE SECOND TASK
          JMS*   =T.F           TERMINATE TASK
CHECK     $DC    0
          LAC    SWITCH         GET SWITCH VALUE
          SZA                   SKIP IF SWITCH NOT SET
          ISZ    CHECK          INDICATE SUCCESS
          JMP*   CHECK          RETURN
```

The call to T.P is given whenever the subroutine CHECK produces a failure return (in the same sense that some system subroutines produce failure returns) to reschedule the call to CHECK. Because tasks are scheduled on a first-in first-out basis, the rescheduled call to CHECK is not executed until each other eligible task in the task queue has been executed.

A task allocates and releases I/O devices by calling appropriate system subroutines, supplying them with "allocation masks." An allocation mask is a representation of the set of I/O devices which are involved in an allocation operation. Each bit position in the mask is associated with one I/O device. If a bit position contains a 1, the corresponding I/O device is involved in the operation; otherwise, it is not. The bit position assignments are given by the following table:

Bit Position	I/O Device
9	201 Dataphone Input
10	201 Dataphone Output
11	Reader
12	Punch
13	Keyboard
14	Teleprinter
15	D/A Converter
16	Push Buttons
17	Display

The following system subroutines have been defined for controlling I/O device allocation:

T.A - The I/O devices specified by the allocation mask in bits 9-17 of the AC are allocated. The calling task is terminated, and the return from this subroutine is scheduled as a task to be executed after the specified devices become available. Bits 0-4 of the AC are ignored.

T.R - The I/O devices specified by the allocation mask in bits 9-17 of the AC are released. Bits 0-4 of the AC are ignored.

In order to guarantee that all scheduled user tasks become eligible for execution in a finite amount of time, I/O device allocation must be performed according to the following rule:

> Whenever an I/O device is allocated, all other I/O devices which are to be allocated before it is released must also be allocated.

As an example of I/O device allocation, consider two tasks, which are scheduled one immediately after the other, whose I/O device allocation activity is summarized by the following tables (where $t_{i,k+1} > t_{i,k}$):

Task #1:

Time	Devices Allocated	Devices Released
t_{11}	A	-
t_{12}	B	-
t_{13}	-	A
t_{14}	C	-
t_{15}	-	B,C

Task #2:

Time	Devices Allocated	Devices Released
t_{21}	C	-
t_{22}	B	-
t_{23}	-	B,C

Assume the rule given above is ignored, and the I/O devices are allocated precisely as shown in the above tables. Then, if $t_{22} > t_{12} > t_{21}$ $t_{14} \rightarrow \infty$ and $t_{22} \rightarrow \infty$ because Task #1 will not release device B until it can allocate device C, and Task #2 will not release device C until it can allocate device B.

By applying the allocation rule to the above tables, the following new tables are obtained:

Task #1:

Time	Devices Allocated	Devices Released
t'_{11}	A,B,C	-
t'_{12}	-	B,C
t'_{13}	B,C	-
t'_{14}	-	A,C
t'_{15}	C	-
t'_{16}	-	B,C

Task #2:

Time	Devices Allocated	Devices Released
t'_{21}	B,C	-
t'_{22}	-	B
t'_{23}	B	-
t'_{24}	-	B,C

With this modification, all tasks will become eligible for execution. (A new task is scheduled and the calling task is terminated each time I/O devices are allocated.)

A subroutine which may be called by several concurrently executing tasks and which allows tasks other than the one which called it to execute before it returns is in danger of being reentered from one task while it is servicing another. This event results in the loss of the return address for the subroutine and perhaps some of the data upon which the subroutine operates. To facilitate the writing of reentrable subroutines (i.e., subroutines which are protected against reentry), the following system subroutines have been defined:

T.L - Lock subroutine against reentry. If the location which immediately follows the call to T.L does not contain zero, the call to the subroutine whose entry point immediately precedes the call to T.L is rescheduled. Otherwise, the content of the location which immediately precedes the call to T.L is copied into the location which immediately follows the call to T.L.

T.U - Unlock reentrable subroutine. The location whose address is the address contained in the word which immediately follows the call to T.U plus 2 is zeroed, and a JMP to the address which was stored in that location before it was zeroed is executed.

Because both T.L and T.U must preserve the contents of the AC and MQ, these subroutines have the following special calling sequences:

Calling sequence for T.L:

```
----    $DC    0              (reentrable subroutine entry
                              point)
        JMS*   =T.L
        $DC    0              (save location for T.L)
        ----                  (return)
```

Calling sequence for T.U:

```
        JMS*   =T.U
        $DC    ----           (subroutine entry point)
```

As an example of the use of T.L and T.U, consider the reentrable subroutine WAIT which returns to its calling task after all tasks on the task queue have had a chance to execute. An algorithm for this subroutine is the following:

```
WAIT    $DC    0
        JMS*   =T.L           SET REENTRY LOCK
        $DC    0              SAVE LOC FOR T.L
```

```
SKP                 SCHEDULE NEXT LOC AS TASK
SKP                 RETURN
JMS*  =T.P          SCHEDULE PREVIOUS LOC AS TASK
JMS*  =T.U          UNLOCK SUBROUTINE & RETURN
$DC   WAIT          SUBROUTINE ENTRY POINT
```

3.3 Format Conversions

Characters are represented internally in the System by 6-bit codes to facilitate storage of three characters per word. Since ASCII character codes must be available for teletype, paper tape, and dataphone I/O, conversions between ASCII and 6-bit codes must be frequently performed. In addition, the 11-bit sign-magnitude coordinates required by the display control's vector mode must often be converted to and from 18-bit two's complement representation. To satisfy these requirements, the following system subroutines have been defined:

C.B6 - The binary number given in the AC is converted to its corresponding 6-bit octal representation, which is returned in the AC and MQ (high-order digits in AC, low-order digits in MQ).

C.6A - The 6-bit code given in bits 12-17 of the AC is converted to the corresponding ASCII code, which is returned in bits 10-17 of the AC, with bits 0-9 cleared and the parity bit of the ASCII code (i.e., bit 10 of the AC) set, regardless of the parity. Bits 0-11 of the AC are ignored on entry.

C.A6 - The ASCII code given in bits 10-17 of the AC is converted to the corresponding 6-bit code, which is returned in bits 12-17 of the AC, with bits 0-11 cleared. Bits 0-9 of the AC and the parity bit of the ASCII code (i.e., bit 10 of the AC) are ignored on entry.

C.CB - The vector mode sign-magnitude display coordinate given in bits 7-17 of the AC is converted to the corresponding two's complement representation, which is returned in the AC. Bits 0-6 of the AC are ignored on entry.

C.BC - The two's complement number in the AC is converted modulo 2^{10} to the corresponding vector mode sign-magnitude display coordinate representation, which is returned in bits 7-17 of the AC with bits 0-6 cleared.

The 6-bit codes used by the System may each be represented by two octal digits as shown by the following table:

First Octal Digit \ Second Octal Digit	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	A	B	C	D	E	F
2	G	H	I	J	K	L	M	N
3	O	P	Q	R	S	T	U	V
4	W	X	Y	Z	*	/	+	-
5	(	)	[	]	<	=	>	↑
6	←	.	,	:	;	?	!	'
7	"	$	#	&	cr	ℓf	sp	

cr ≡ carriage return
ℓf ≡ line feed
sp ≡ space

All ASCII characters which do not appear in the table are mapped into 77_8 . The only printing characters which are treated in this manner are "%", "@", and " ".

3.4 Buffered I/O

Input data from the dataphone, the paper tape reader, and the keyboard, as well as output data to the dataphone,

paper tape punch, and teleprinter, are buffered by the System. In the event that an input buffer is empty or an output buffer is full and the system subroutine which transfers data between the buffer and a task is called, the return from the subroutine is scheduled as a task to be executed only after the state of the buffer changes, and execution of the calling task is terminated.

3.4.1 Dataphone I/O

The following system subroutines have been defined for managing the 201 dataphone buffers:

B.FI* - An image is fetched from the 201 dataphone input buffer and is returned in bits 10-17 of the AC. Bits 0-9 of the AC are cleared, unless the image is an end-of-record character in which case bits 0-4 are set and bits 5-9 are cleared. A failure return is made if the data set is not connected.

B.FO* - The image in bits 10-17 of the AC is sent to the 201 dataphone output buffer. If bit 0 of the AC is set, the image is interpreted as an end-of-record character, and transmission is begun. A failure return is made before the image is buffered if the data set is not connected.

Since actual dataphone transmission is record-oriented (although transfer of data between the dataphone buffers and tasks is not), the return from B.FI to the calling task is delayed until the dataphone input buffer contains a complete record, and the return from B.FO is delayed until the last record transmitted has been affirmatively acknowledged by the other party. In simpler terms, the dataphone input buffer is considered to be empty whenever it does not contain a complete record, and the dataphone output buffer is considered to be full whenever the last transmitted record has not been affirmatively acknowledged.

Dataphone records are formatted according to the conventions adopted by The University of Michigan Computing Center at the time of this report. Each record is formatted (if transmitted) or interpreted (if received) by the System and consists of the following sections:

1. Several synchronous idle (SYN) characters (026_8). (At least two are required when receiving; eight are transmitted.)

2. A data link escape (DLE) character (220_8).

3. Data. The 8-bit images in this section are arbitrary binary, with the exception that a DLE character (with either parity) is preceded by a DLE. The first DLE is ignored when the record is received, and serves only to cause the second one to be interpreted as data. (A pair of characters consisting of a DLE followed by a SYN is ignored when receiving, although this sequence is never transmitted.)

4. A DLE character.

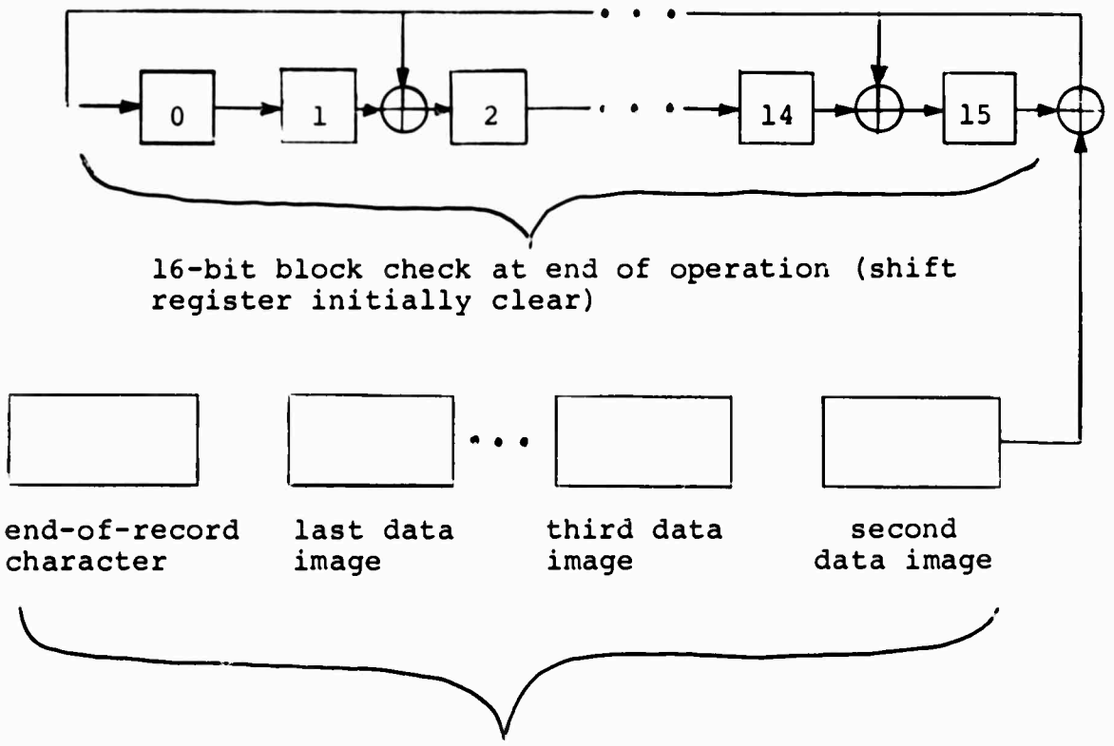
5. An end-of-record character.

6. The high-order 8 bits of the block check (described below).

7. The low-order 8 bits of the block check (described below).

8. A pad character (377_8).

In order to facilitate detection of burst errors, a 16-bit cyclic block check is included in each dataphone record. For purposes of computing this block check, the data sequence (consisting of the concatenation of the second through the last data images, plus the end-of-record character) is regarded as a cyclic polynomial code. The block check is obtained by simultaneously multiplying the polynomial representation of the data sequence by X^{16} and dividing it by $X^{16} + X^{15} + X^2 + 1$ (where the coefficients of the polynomials are taken from the field of two elements). The following diagram illustrates this operation:



Whenever a dataphone record is received by either party, the block check is computed and compared with the received block check. If the two block checks match, the dataphone record is assumed to have been received correctly, and an affirmative acknowledgment is returned when the receiving party is ready for the next record. However, if the two block checks do not match, a negative acknowledgment, which is a request for the record to be retransmitted, is returned, and the incorrectly received record is discarded. The System assumes complete responsibility for managing acknowledgments and retransmissions for the 339.

Whenever a dataphone record is received with a correct block check, the first data image is examined. If it is zero, user tasks are given access to it via the system subroutine B.FI. Otherwise, a special 201-to-teleprinter task is scheduled to type the record (interpreting it as a sequence of ASCII codes) as soon as the teleprinter becomes available. In

this way, unsolicited messages from the remote party are typed and routed clear of tasks which are using the dataphone.

Whenever the end-of-record character for either a transmitted or received record is an enquiry (005_8) or an end-of-transmission (204_8), both dataphone buffers (input and output) are cleared, and the last record transmitted is considered to have been affirmatively acknowledged. Note that transmitted records of this form will be processed normally by the System (except that immediate acknowledgment will be assumed), but received records of this form will be discarded once the end-of-record character is detected.

As an example of the use of B.FI and B.FO, consider the task MIRROR which receives 64 dataphone images in an arbitrary number of records (not including the zero images required to route records to tasks), transmits all of them in one dataphone output record, and ignores the remainder of the last dataphone input record which it examined. An algorithm for this task is the following (L.T is described in Section 3.11):

```
MIRROR    LAW     600       GET ALLOCATION MASK
          JMS*    =T.A      ALLOCATE 201 INPUT & OUTPUT
          LAW     17700     LOAD AC WITH -64
          DAC     COUNT     INITIALIZE IMAGE COUNT
START     JMS*    =B.FI     GET REDUNDANT IMAGE
          JMP     HELP      DATA SET NOT CONNECTED
READ      JMS*    =B.FI     GET INPUT IMAGE
          JMP     HELP      DATA SET NOT CONNECTED
          SPA               SKIP IF NOT END OF RECORD
          JMP     START     READ NEXT RECORD
          JMS*    =B.FO     PUT IN OUTPUT BUFFER
          JMP     HELP      DATA SET NOT CONNECTED
          ISZ     COUNT     SKIP IF RECORD LONG ENOUGH
          JMP     READ      READ NEXT IMAGE
          JMS*    =B.FI     GET INPUT IMAGE
          JMP     HELP      DATA SET NOT CONNECTED
```

```
        SMA             SKIP IF END OF RECORD
        JMP     *-3     READ ANOTHER IMAGE
        JMS*    =B.FO   TERMINATE OUTPUT RECORD
        JMP     HELP    DATA SET NOT CONNECTED
        LAW     600     GET ALLOCATION MASK
        JMS*    =T.R    RELEASE 201 INPUT & OUTPUT
        JMS*    =T.F    TERMINATE TASK
HELP    LAW     600     GET ALLOCATION MASK
        JMS*    =T.R    RELEASE 201 INPUT & OUTPUT
        LAW     10      GET ALLOCATION MASK
        JMS*    =T.A    ALLOCATE TELEPRINTER
        LAC     =TEXT   GET ADDRESS OF TEXT LIST
        JMS*    =L.T    TYPE TEXT LIST
        LAW     10      GET ALLOCATION MASK
        JMS*    =T.R    RELEASE TELEPRINTER
        JMS*    =T.F    TERMINATE TASK
TEXT    $DC     20
        $TEXT   ' ATA SET NOT CONNECTED.  'MIRROR' TERMINATED."
        $DC     747577
```

3.4.2 Paper Tape I/O

The following system subroutines have been defined for managing the paper tape reader and punch buffers:

B.R* - An image is fetched from the reader buffer and returned in bits 10-17 of the AC. Bits 0-9 of the AC are cleared. Only one end-of-record character (zero) may be returned by two successive calls to B.R. A failure return is made if the reader is out of tape and the reader buffer is empty.

B.P* - The image in bits 10-17 in the AC is sent to the punch buffer. A failure return is made if the punch is out of tape and the punch buffer is full.

Paper tape formats are arbitrary, subject to the restriction that a zero image (i.e., a line of blank tape) which immediately follows a nonzero image is interpreted as an end-of-record character and all other zero images are ignored. However, the format which is read and punched by the data transfers of the idle-time task (Section 4.1) is recommended for compatibility reasons. In this format, the two high-order bits of each 8-bit tape image are interpreted as control information, and the remaining 6 bits are interpreted as data. The two control bits are interpreted as follows:

00	mode change
01	binary origin
10	binary data
11	alphanumeric data

There are 64 possible mode changes (designated by the low-order 6 bits of a mode change tape image), only one of which has been defined at the time of this writing, i.e., the end-of-record character 000_8. (An example of possible future mode change assignments is a set of relocation modes for relocatable binary records.)

A binary block consists of three binary origin images followed by a multiple of three binary data images. The block represents a set of 18-bit words to be loaded starting at the address indicated by the data bits of the three origin images. For example, the binary block which indicates that location 23572_8 should contain 621365_8 and that location 23573_8 should contain 176234_8 is the following:

```
102
135      origin 23572
172
262
213      data 621365
265
217
262      data 176234
234
```

A binary record is a concatenation of binary blocks, followed by the end-of-record character (000_8).

An alphanumeric record consists of an arbitrary number of alphanumeric tape images (where the 6 data bits in each image represent a 6-bit character code), followed by an end-of-record character (000_8).

As an example of the use of the paper tape I/O system subroutines, consider a task COPY which copies one record of paper tape:

```
COPY    LAW     140     GET ALLOCATION MASK
        JMS*    =T.A    ALLOCATE READER & PUNCH
        JMS*    =B.R    GET IMAGE FROM READER
        JMP     RERR    READER OUT OF TAPE
        SNA             SKIP IF NOT END OF RECORD
        JMP     *+4     END OF RECORD
        JMS*    =B.P    PUNCH IMAGE
        JMP     PERR    PUNCH OUT OF TAPE
        JMP     COPY+2  READ NEXT IMAGE
        JMS*    =B.P    PUNCH END OF RECORD
        JMP     PERR    PUNCH OUT OF TAPE
        LAW     140     GET ALLOCATION MASK
        JMS*    =T.R    RELEASE READER & PUNCH
        JMS*    =T.F    TERMINATE TASK
RERR    LAC     =RERRT  GET ADDRESS OF TEXT LIST
        SKP             TYPE DIAGNOSTIC
PERR    LAC     =PERRT  GET ADDRESS OF TEXT LIST
        DAC     TEXT    SAVE ADDRESS OF TEXT LIST
        LAW     140     GET ALLOCATION MASK
        JMS*    =T.R    RELEASE READER & PUNCH
        LAW     10      GET ALLOCATION MASK
        JMS*    =T.A    ALLOCATE TELEPRINTER
        LAC     TEXT    GET ADDRESS OF TEXT LIST
        JMS*    =L.T    TYPE DIAGNOSTIC
        LAC     =END    GET ADDRESS OF TEXT LIST
```

```
        JMS*    =L.T        TYPE TEXT LIST
        LAW     10          GET ALLOCATION MASK
        JMS*    =T.R        RELEASE TELEPRINTER
        JMS*    =T.F        TERMINATE TASK
RERRT   $DC     2
        $TEXT   "READER"
PERRT   $DC     2
        $TEXT   "PUNCH"
END     $DC     15
        $TEXT   "OUT OF TAPE"
        $DC     747577
        $TEXT   "COPY TASK TERMINATED"
        $DC     747577
```

3.4.3 Teletype I/O

The following system subroutines have been defined for managing the keyboard and teleprinter buffers:

B.K - A 6-bit character is fetched from the keyboard buffer and returned in bits 12-17 of the AC. Bits 0-11 of the AC are cleared.

B.T - The three six-bit characters in bits 0-5, 6-11, and 12-17 of the AC are sent to the teleprinter buffer to be typed in respective order. (The null character 77_8 will not be typed, even as a non-printing character.)

As an example of the use of these subroutines, consider the task ENCODE which accepts characters from the keyboard and types the octal representation of the corresponding 6-bit codes. When a null character is typed, the task is terminated. An algorithm for this task is the following:

```
ENCODE  LAW     30          GET ALLOCATION MASK
        JMS*    =T.A        ALLOCATE KEYBOARD & TELEPRINTER
        JMS*    =B.K        GET CHARACTER FROM KEYBOARD
        SAD     =77         SKIP IF NOT NULL CHARACTER
        JMP     END         TERMINATE TASK
```

```
        JMS*    =C.B6       CONVERT TO 6-BIT OCTAL CODE
        LACQ                GET LOW-ORDER DIGITS
        XOR     =770000     REMOVE HIGH-ORDER ZERO
        JMS*    =B.T        TYPE ENCODED CHARACTER
        LAW     17475       GET CARRIAGE RETURN, LINE FEED CODE
        JMS*    =B.T        TYPE CARRIAGE RETURN, LINE FEED
        JMP     ENCODE+2    PROCESS NEXT CHARACTER
END     LAW     30          GET ALLOCATION MASK
        JMS*    =T.R        RELEASE KEYBOARD & TELEPRINTER
        JMS*    =T.F        TERMINATE TASK
```

3.5 Nonbuffered I/O

Three devices which might appear to require buffering are not buffered: the clock, the A/D converter, and the D/A converter. The clock, which is normally used in an interactive system to check for the occurrence of certain events within specified time intervals, is often programmed in a multiprogramming system such that any task may use it at any time. This is accomplished through the use of a buffer into which entries (each consisting of a return pointer and a time interval) may be inserted at arbitrary points. Since the buffer required is considerably more complicated than those used by other devices, the cost of programming the clock in this manner was found to be excessive.

Since A/D converter data should be interpreted in real time, these data are not buffered. Instead, whenever a task calls the system subroutine to obtain data from the A/D converter, the device is selected, the return from the subroutine is scheduled as a task to be executed after the conversion is complete, and execution of the calling task is terminated.

The D/A converter requires only two microseconds to produce an output after it is selected, whereas the minimum time between selections of a particular D/A channel is four microseconds. Consequently, the System does not buffer D/A converter data.

The following system subroutines have been defined for nonbuffered I/O:

N.C - Execution of the calling task is terminated and the return from N.C is scheduled as a task to be executed at least the number of sixtieths of a second later which is the two's complement of the number given in the AC.

N.A - The channel of the A/D converter specified in bits 12-17 of the AC is selected, and the converted value, when obtained, is returned in bits 0-11 of the AC. Bits 12-17 of the AC are cleared. The returned value, if interpreted as an ordinary two's complement number, is $-2^{17}(1+V/5)$, where V is the applied input voltage (which ranges from 0 to -10 volts).

N.D1- D/A converter channel #1 is selected. The output of the channel is set to $-5(1+2^{-17}A)$ volts, where A is the content of the AC.

N.D2- D/A converter channel #2 is selected. The output of the channel is set to $-5(1+2^{-17}A)$ volts, where A is the content of the AC.

N.D3- D/A converter channel #3 is selected. The output of the channel is set to $-5(1+2^{-17}A)$ volts, where A is the content of the AC.

As an example of a use of N.C, consider the task PROMPT which types "PLEASE TYPE NOW" once about every eight seconds until the operator types something on the keyboard, and types "THANK YOU" when the operator finishes typing a line. An algorithm for this task is the following:

```
PROMPT    LAW     30         GET ALLOCATION MASK
          JMS*    =T.A       ALLOCATE KEYBOARD & TELEPRINTER
          DZM     DONE       INDICATE NO KEYBOARD RESPONSE
          LAC     =POLITE    GET ADDRESS OF KEYBOARD CHECKER
```

```
        JMS*    =T.S        SCHEDULE KEYBOARD CHECKER
        LAC     =TXT1       GET ADDRESS OF TEXT LIST
        JMS*    =L.T        TYPE "PLEASE TYPE NOW"
        LAW     -1000       GET TIME PARAMETER
        JMS*    =N.C        WAIT ABOUT 8 SECONDS
        LAC     DONE        GET KEYBOARD RESPONSE SWITCH
        SNA                 SKIP IF RESPONSE OBTAINED
        JMP     PROMPT+5    PROMPT OPERATOR  AGAIN
        JMS*    =T.F        TERMINATE EXECUTION
POLITE  JMS*    =B.K        GET KEYBOARD CHARACTER
        XOR     =777700     PRECEDE WITH NULL CHARACTERS
        DAC     DONE        SET KEYBOARD RESPONSE SWITCH
        SAD     =777774     SKIP IF NOT CARRIAGE RETURN
        JMP     *+3         END OF INPUT LINE
        JMS*    =B.T        ECHO CHARACTER ON TELEPRINTER
        JMP     POLITE      GET ANOTHER CHARACTER
        LAC     =TXT2       GET ADDRESS OF TEXT LIST
        JMS*    =L.T        TYPE "THANK YOU"
        LAW     30          GET ALLOCATION MASK
        JMS*    =T.R        RELEASE KEYBOARD AND TELEPRINTER
        JMS*    =T.F        TERMINATE EXECUTION
TXT1    $DC     6
        $TEXT   "PLEASE TYPE NOW"
        $DC     747577
TXT2    $DC     5
        $DC     747577
        $TEXT   "THANK YOU"
        $DC     747577
```

As an example of the use of N.A, consider the task COMPAR which samples channels 0 and 1 of the A/D converter until the inputs on the two channels are close enough to each other that the same value is read from each channel. When this condition is satisfied, the comment "ANALOG INPUTS MATCH" is typed on the teletype. An algorithm for this task is the following:

```
COMPAR    CLA              GET CHANNEL 0 PARAMETER
          JMS*   =N.A      CONVERT CHANNEL 0 VALUE
          DAC    VALUE     SAVE CHANNEL 0 VALUE
          LAW    1         GET CHANNEL 1 PARAMETER
          JMS*   =N.A      CONVERT CHANNEL 1 VALUE
          CMA              FORM 1 COMPLEMENT
          TAD    VALUE     ADD CHANNEL 0 VALUE
          CMA              FORM DIFFERENCE IN VALUES
          SZA              SKIP IF VALUES EQUAL
          JMP    COMPAR    OBTAIN NEW SAMPLES
          LAW    10        GET ALLOCATION MASK
          JMS*   =T.A      ALLOCATE TELEPRINTER
          LAC    FOUND     GET ADDRESS OF TEXT LIST
          JMS*   =L.T      TYPE "ANALOG INPUTS MATCH"
          LAW    10        GET ALLOCATION MASK
          JMS*   =T.R      RELEASE TELEPRINTER
          JMS*   =T.F      TERMINATE EXECUTION
FOUND     $DC    10
          $TEXT  "ANALOG INPUTS MATCH"
          $DC    747577
```

3.6 Push-Button Processing

The following system subroutines have been defined for managing the push buttons which are associated with the display control:

P.T - The task whose address is given in bits 3-17 of the AC is declared to be the service task for manual operation of the push buttons (i.e., this task is scheduled whenever the state of the push buttons is altered by the operator). If the AC contains zero when P.T is called, a null service task (i.e., one which calls P.E and terminates) is used.

P.E - Manual operation of the push buttons is enabled (i.e., the state of the push buttons may be changed by the operator).

P.D - Manual operation of the push buttons is disabled (i.e., the state of the push buttons may not be changed by the operator). A call to P.D is effected whenever the operator changes the state of the push buttons.

P.R - Push buttons 0-11 are read into bits 6-17 of the AC, and bits 0-5 of the AC are cleared.

P.S - Push buttons 0-11 are set according to bits 6-17 of the AC.

As an example of the use of these subroutines, consider the task BUTTON which enables manual operation of the push buttons and sets the button numbered one greater (modulo 12) than the number of the one pushed by the operator. The procedure is terminated and all push buttons are cleared when a keyboard character is struck. An algorithm for this task is the following:

```
BUTTON    LAW    22        GET ALLOCATION MASK
          JMS*   =T.A      ALLOCATE KEYBOARD & PUSH BUTTONS
          LAC    =SERV     GET ADDRESS OF SERVICE TASK
          JMS*   =P.T      DECLARE SERVICE TASK
          CLA              GET INITIAL PUSH BUTTON STATE
          DAC    STATE     SAVE FOR USE BY SERV
          JMS*   =P.S      SET INITIAL PUSH BUTTON STATE
          JMS*   =P.E      ENABLE MANUAL OPERATION
          JMS*   =B.K      GET KEYBOARD CHARACTER
          JMS*   =P.D      DISABLE MANUAL OPERATION
          CLA              GET FINAL PUSH BUTTON STATE
          JMS*   =P.S      CLEAR PUSH BUTTONS
          CLA              GET NULL SERVICE PARAMETER
          JMS*   =P.T      DECLARE NULL SERVICE TASK
          LAW    22        GET ALLOCATION MASK
          JMS*   =T.R      RELEASE KEYBOARD & PUSH BUTTONS
          JMS*   =T.F      TERMINATE TASK
SERV      JMS*   =P.R      READ PUSH BUTTONS
```

```
XOR     STATE       ISOLATE LAST BUTTON PUSHED
RCR                 FORM MASK FOR SETTING BUTTONS
SNA                 SKIP IF NOT BUTTON #11
LAC     =4000       SET BUTTON #0 BIT
DAC     STATE       SAVE NEW PUSH BUTTON STATE
JMS*    =P.S        SET NEW PUSH BUTTON STATE
JMS*    =P.E        ENABLE MANUAL OPERATION
JMS*    =T.E        TERMINATE TASK
```

3.7 Display Control Communication

The following system subroutines have been defined for communicating with the display control:

D.E - Display interrupts are enabled (i.e., a light pen flag interrupt or an internal stop interrupt will cause the System to read the display status information required for D.A, D.Y, D.X, and D.O and to schedule the appropriate service task).

D.D - Display interrupts are disabled (i.e., the System will ignore light pen flag and internal stop interrupts). A call to D.D is effected whenever a display interrupt occurs.

D.P - The task whose address is given in bits 3-17 of the AC is declared to be the service task for light pen flags. This task is scheduled whenever the light pen sees an intensified portion of the display on which the light pen is enabled (see Section 3.10), providing that display interrupts are enabled (via D.E). If the AC contains zero when D.P is called, a null service task (i.e., one which calls D.E and terminates) is used.

D.A - The address of the display on the last display interrupt is returned in bits 3-17 of the AC with bits 0-2 clear.

D.Y - The y coordinate of the display (measured relative to the center of the screen in scale x1) on the last display interrupt is returned in the AC as a two's complement number.

D.X - The x coordinate of the display (measured relative to the center of the screen in scale x1) on the last display interrupt is returned in the AC as a two's complement number.

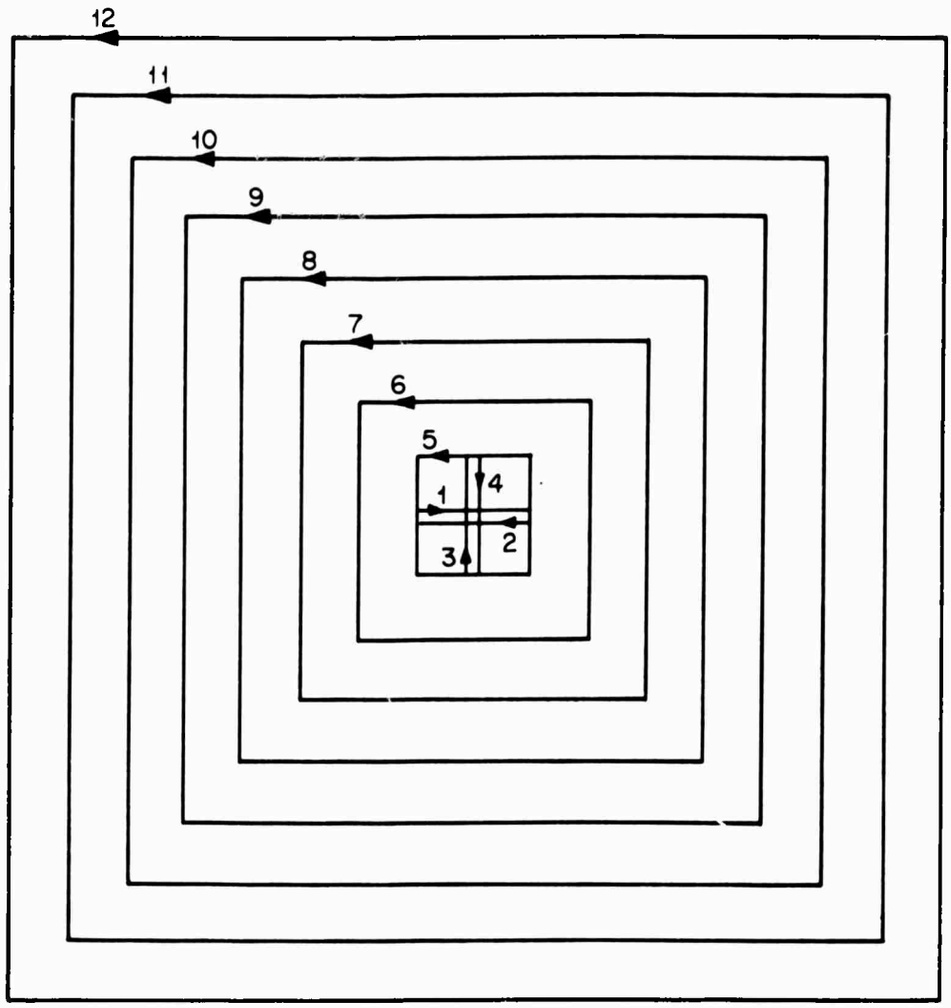
D.O*- The address which is the operand of the push jump instruction which was the number of entries given in bits 12-17 of the AC above the last entry in the display control's push-down list on the last display interrupt is returned in bits 3-17 of the AC with bits 0-2 clear. (A more meaningful interpretation of this subroutine may be obtained from the examples in Section 3.10.) A failure return is made if the indicated push jump instruction does not exist.

The external stop interrupt and the edge flag interrupt are not used. The function of the external stop interrupt may be performed via an unconditional internal stop interrupt (via S.LU, which is described in Section 3.10). Since the virtual display area established by the System is 75 inches by 75 inches, the edge flags, if used, would occur on the left and lower edges of the screen, but not on the upper or right edges. Because of this inconsistency, the edge flags are not used.

3.8 Light Pen Tracking

A light pen tracking algorithm is supplied with the System to enable user tasks to follow the motion of the light pen. This algorithm has been empirically determined to track the light pen at any attainable speed, and it is insensitive to changes in direction because it does not involve prediction.

The tracking algorithm may be described with the aid of the following diagram:



When the display for the tracking algorithm is begun, strokes 1 and 2 are drawn. (Strokes 1 and 2 are actually coincident.) The x coordinate of the first light pen hit on each stroke is recorded. If both x coordinates are obtained, a new x coordinate for the tracking cross is computed as their average. Strokes 3 and 4 are then drawn, and a new y coordinate for the tracking cross is computed in similar manner if both y coordinates are obtained.

If any one of the four coordinates required to compute a new position of the tracking cross is not obtained, a search pattern consisting of concentric squares 5 through 12 is drawn. When a light pen hit is detected on any one of these squares, the search pattern is terminated, and the tracking cross is placed at the coordinates of the hit. If square 12 is completed and no light pen hit is detected, the tracking process is terminated.

Whenever the tracking cross is positioned via the search pattern, rather than by averaging coordinates, the tracking display is immediately repeated. The remainder of the active display structure (Section 3.9) is not displayed until the tracking cross can be positioned by averaging coordinates. In this way, the tracking display is given priority over all other displays whenever the light pen is being moved rapidly and tracking is in process.

The following system subroutines have been defined for controlling the tracking process:

X.I - The tracking cross is placed at the y coordinate given in the AC and the x coordinate given in the MQ, and the tracking process is begun. The coordinates, which are given as two's complement numbers, are interpreted modulo 2^{10} measured in scale x1 relative to the center of the screen.

X.R - The tracking process is resumed with the tracking cross at the coordinates where tracking was last terminated (by X.T or by completion of square 12).

X.T - The tracking process is terminated. (The tracking cross is removed from the screen.)

X.S*- A failure return is made if tracking is in process.

X.Y - The y tracking coordinate is returned in the AC as a two's complement number measured in scale x1 relative to the center of the screen. If tracking

is not in process, the y coordinate where tracking was last terminated is returned.

X.X - The x tracking coordinate is returned in the AC as a two's complement number measured in scale x1 relative to the center of the screen. If tracking is not in process, the x coordinate where tracking was last terminated is returned.

The tracking algorithm is independent of D.E and D.D.

3.9 Display Structure Topology

Each entity to be displayed is represented in the display structure provided by the System as a position in the hierarchy of the entities which constitute the picture. Each position in the hierarchy is implemented as a display subroutine which is called a level. A level which is being executed by the display control at least once on every frame is called an active level. One particular level, which is always active and is an integral part of the system, represents the 75 inch by 75 inch virtual display area of the display control and is called the highest active level.

A display subroutine which is not itself a level and which contains no calls to levels is called a leaf. All of the drawing of visible portions of the picture is accomplished by leaves. A leaf is subject to the restriction that the state of the display (coordinates, light pen status, scale, intensity, blink status, light pen sense indicator) must be the same when the subroutine returns as when it is entered. Consequently, because the display control's POP instruction does not restore coordinates, the only data modes which are useful in leaves are vector mode, short vector mode, and increment mode.

The set L of all levels and leaves (both active and non-active) is partially ordered, i.e., there exists a relation "$\leq$" defined on L such that

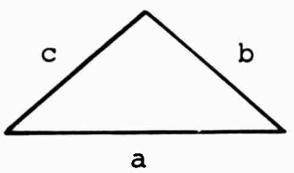
(1) $\forall x \in L$ $\quad x \leq x$

(2) $\forall x, y \in L$ $\quad x \leq y$ and $y \leq x \Rightarrow x = y$

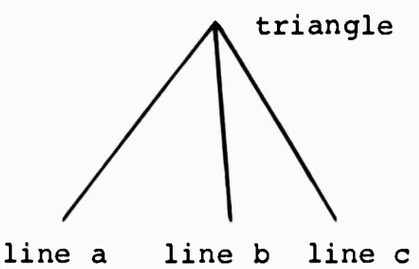
(3) $\forall x, y, z \in L$ $\quad x \leq y$ and $y \leq z \Rightarrow x \leq z$

The semantic interpretation of the expression $x \leq y$ is that any modification of the entity represented by the level x (or in the drawing produced by the leaf x, if x is a leaf) will effect a corresponding modification in the entity represented by the level y. When $x \leq y$, the level y is said to <u>own</u> the level or leaf x. An <u>attribute</u> of a level y is a level or leaf x such that $x \leq y$ and there does not exist a level z different from x and y such that $x \leq z$ and $z \leq y$.

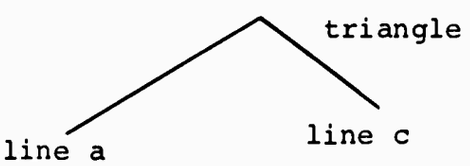
As an example of this interpretation of the relation "$\leq$", consider a triangle which is to be represented internally as a set of three lines:



A display structure for this triangle may be represented by the following diagram. (In the diagram, $x \leq y$ is represented by a line joining x and y such that y appears above x in the diagram.)



Note from the diagram that the triangle owns each of its sides (lines a, b, and c). If line b is now deleted, the display structure assumes the following form:



The triangle is obviously modified by this operation (in fact, it is no longer a triangle). However, the fact that the triangle has been modified does not imply that all of its attributes have been modified. In this example, lines a and c remain unchanged.

The set X of all active levels and the leaves which they own is also partially ordered, since $X \subset L$ and L is partially ordered. Because the highest active level represents the virtual display area of the display control, it owns every element of X. Consequently, if the operator "+" is defined by the conditions

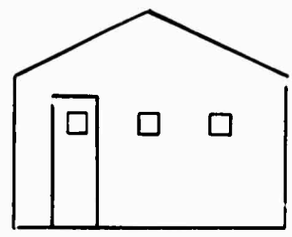
(1) $\forall x, y \in X \qquad x + y \in X$

(2) $\forall x, y \in X \qquad x \leq x + y \text{ and } y \leq x + y$

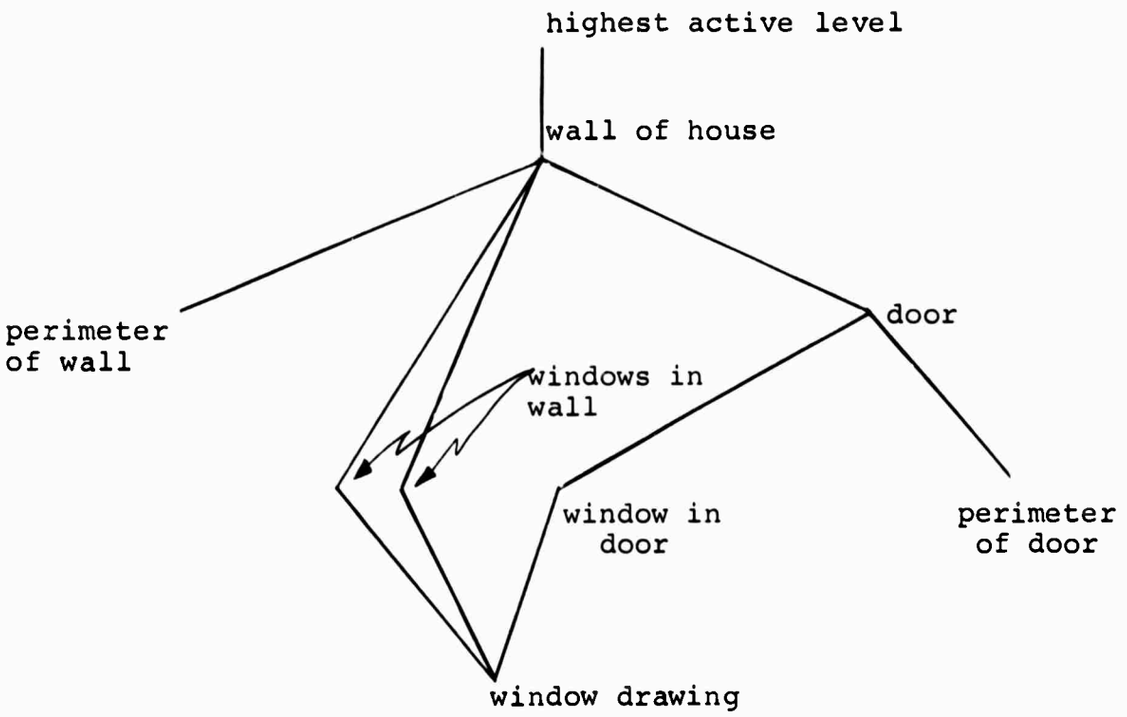
and (3) $\forall x, y, z \in X \qquad x \leq z \text{ and } y \leq z \Longrightarrow x + y \leq z$,

the pair (X,+) is a semilattice. The semantic interpretation of the expression x+y is that x+y is a level which represents the most primitive entity which owns both of the entities represented by the levels x and y .

As an example of the interpretation of the operator "+", consider the following drawing of one exterior wall of a house:

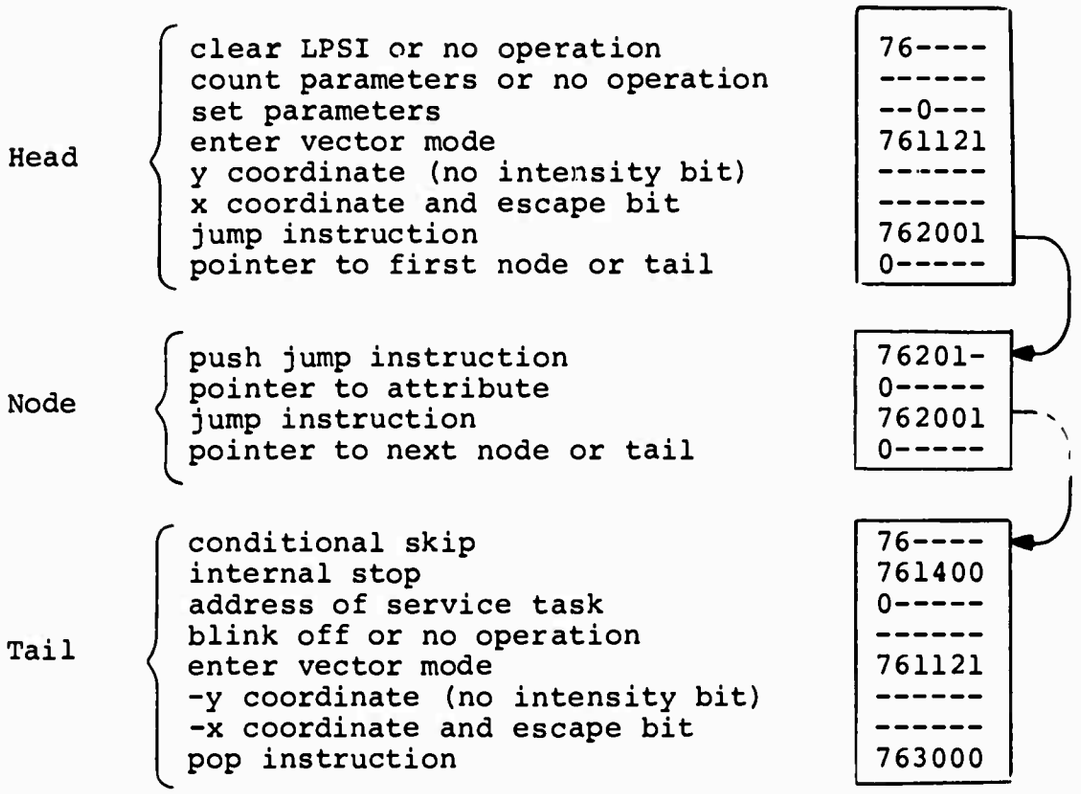


For purposes of illustration, assume that all three windows in the picture are identical, each instance of each entity in the drawing is represented by a separate level, and the drawing shown is the only one being displayed. The display structure, then, assumes the following form:



Assume that a task which records two references to the picture with the light pen is being executed, and that the most primitive entity which owns both items referenced is to be deleted. Clearly, the portion of the structure which should be removed consists of everything which x+y owns, where x and y are the two levels which represent the entities referenced with the light pen. For example, if the door perimeter and a window in the wall of the house are referenced, the entire wall of the house is deleted, but if the door perimeter and the window in the door are referenced, only the door is deleted.

A level is implemented as the data structure shown by the following diagram (all numbers are octal):



The following system subroutines have been defined for managing the display structure topology. (Examples of their use are given in Section 3.10.)

S.TL*- A level is created and its address (i.e., the address of the first location in its head) is returned in bits 3-17 of the AC with bits 0-2 clear. A failure return is made if the level cannot be created because of insufficient free display storage.

S.TD*- The non-active level whose address is given in bits 3-17 of the AC is destroyed. A failure return is made if the level has attributes.

S.TI*- The level or leaf whose address is given in bits 3-17 of the MQ is inserted into (i.e., made an attribute of) the level whose address is given in bits 3-17 of the AC. The created node is inserted immediately after the head

in the level data structure. A failure return is made if the required node cannot be created because of insufficient free display storage.

S.TR*- The attribute whose address is given in bits 3-17 of the MQ is removed from the level whose address is given in bits 3-17 of the AC. This subroutine does not return until the display control has completed the current frame. (Tasks other than the calling task are executed during this delay.) A failure return is made if the specified attribute is not found in the specified level.

3.10 Level Modification

The following system subroutines have been defined for modifying existing levels:

S.LH - The address of the highest active level is returned in bits 3-17 of the AC with bits 0-2 clear.

S.LY - The y coordinate of the level whose address is given in bits 3-17 of the AC is set to the value given in the MQ. The given coordinate is interpreted as a two's complement number in the scale of the specified level, measured relative to the y coordinate of each level of which the specified level is an attribute. This subroutine has no effect on the highest active level, where the coordinates are at the center of the screen.

S.LX - The x coordinate of the level whose address is given in bits 3-17 of the AC is set to the value given in the MQ. The given coordinate is interpreted as a two's complement number in

the scale of the specified level, measured relative to the x coordinate of each level of which the specified level is an attribute. This subroutine has no effect on the highest active level, where the coordinates are at the center of the screen.

S.LP - The scale, intensity, and light pen status are set on the level whose address is given in bits 3-17 of the AC according to bits 9-17 of the MQ. The content of the MQ is interpreted as follows:

Bits	Interpretation
9	set scale according to bits 10-11
10-11	n, where scale is $x2^n$
12	set light pen status according to bit 13
13	light pen status (1 = enabled, 0 = disabled)
14	set intensity according to bits 15-17
15-17	intensity value

This subroutine has no effect on the highest active level, where the scale is x8, the intensity is 7, and the light pen is disabled.

S.LBE- The displays generated by calls (either direct or indirect) to leaves from the level whose address is given in bits 3-17 of the AC are caused to blink with a 0.5-second period. Because the 339 POP instruction does not restore the blink status, care must be taken to insure that this blink is not simultaneously effective on any level of which the given level is an owner. This subroutine has no effect on the highest active level, where blink is disabled.

S.LBD- Blinking of the level whose address is given in bits 3-17 of the AC is disabled (i.e., the effect of a call to S.LBE is removed).

S.LC - The scale and/or intensity is counted up or down one unit on the level whose address is given in bits 3-17 of the AC according to bits 12-15 of the MQ, which are interpreted as follows:

Bit	Interpretation
12	Count scale according to bit 13
13	1 = multiply scale by 2, 0 = divide scale by 2
14	Count intensity according to bit 15
15	1 = increment intensity by unity, 0 = decrement intensity by unity.

This subroutine has no effect on the highest active level.

S.LU - An unconditional scheduling of the task whose address is given in bits 3-17 of the MQ is effected whenever display interrupts are enabled (via D.E) and the tail of the level whose address is given in bits 3-17 of the AC is executed. This subroutine has no effect on the highest active level.

S.LS - The task whose address is given in bits 3-17 of the MQ is scheduled whenever display interrupts are enabled (via D.E), the tail of the level whose address is given in bits 3-17 of the AC is executed, and the coordinates of that level are on the screen. This subroutine has no effect on the highest active level.

S.LL - The task whose address is given in bits 3-17 of the MQ is scheduled whenever display interrupts are enabled (via D.E), the tail of the level whose address is given in bits 3-17 of the AC is executed, and the light pen sense indicator has been set during execution of that level. This subroutine has no effect on the highest active level.

S.LN - The effect of S.LU, S.LS, or S.LL is removed from the level whose address is given in bits 3-17 of the AC.

Whenever the scale, light pen status, intensity, blink status, or coordinates are not set on a level, the quantities which are not set on that level are the same as those on the level of which it is an attribute.

Some user subroutines which call these system subroutines, as well as those in Section 3.9, are given below. LVL generates a level, inserts a specified attribute into it, sets the x and y coordinates and display parameters on the generated level, and inserts the generated level into a specified owner level. BUTN calls on LVL, and then establishes a task to be scheduled whenever the light pen sense indicator is set while the display control is executing the generated level. BUTX generates a text leaf from a specified text list, and then calls on BUTN, using the generated text leaf as the attribute parameter. CHEW (which calls on ATTR to find the first attribute of a level) destroys a given display structure, and salvages all storage from the destroyed levels and text leaves. The display structure on which CHEW operates must satisfy two conditions:

(1) It must assume the form of a semilattice.

(2) The maximum element of the display structure must not be owned by any level (other than itself, if it itself is a level). (L.D and L.L are described in Section 3.11.)

```
*CALLING SEQUENCE:
*         JMS    LVL
*         $DC    ----      (LOC CONTAINING POINTER TO OWNER)
*         $DC    ----      (Y COORDINATE)
*         $DC    ----      (X COORDINATE)
*         $DC    ----      (DISPLAY PARAMETER)
*         ----             (RETURN IF DISPLAY STORAGE EXCEEDED)
*         ----             (RETURN)
```

```
*AC CONTENT  ON ENTRY:
*          POINTER TO ATTRIBUTE
*AC CONTENT  ON RETURN:
*          POINTER TO CREATED LEVEL
LVL        $DC     0
           JMS*    =T.L        SET REENTRY LOCK
           $DC     0
           DAC     LVL4        SAVE POINTER TO ATTRIBUTE
           JMS*    =S.TL       CREATE A LEVEL
           JMP     LVL3        DISPLAY STORAGE EXCEEDED
           DAC     LVL5        SAVE POINTER TO LEVEL
           LAC     LVL4        GET POINTER TO ATTRIBUTE
           LMQ                 SET UP PARAMETER
           LAC     LVL5        GET POINTER TO LEVEL
           JMS*    =S.TI       INSERT ATTRIBUTE
           JMP     LVL2        DISPLAY STORAGE EXCEEDED
           LAC*    LVL+2       GET FIRST PARAMETER
           DAC     LVL4        SAVE FIRST PARAMETER
           ISZ     LVL+2       ADVANCE TO NEXT PARAMETER
           LAC*    LVL+2       GET Y COORDINATE
           LMQ                 SET UP PARAMETER
           LAC     LVL5        GET POINTER TO LEVE
           JMS*    =S.LY       SET Y COORDINATE
           ISZ     LVL+2       ADVANCE TO NEXT PARAMETER
           LAC*    LVL+2       GET X COORDINATE
           LMQ                 SET UP PARAMETER
           LAC     LVL5        GET POINTER TO LEVEL
           JMS*    =S.LX       SET X COORDINATE
           ISZ     LVL+2       ADVANCE TO NEXT PARAMETER
           LAC*    LVL+2       GET DISPLAY PARAMETER
           LMQ                 SET UP PARAMETER
           LAC     LVL5        GET POINTER TO LEVEL
           JMS*    =S.LP       SET DISPLAY PARAMETER
           LAC     LVL5        GET POINTER TO LEVEL
           LMQ                 SET UP PARAMETER
```

```
            LAC*    LVL4        GET POINTER TO OWNER
            JMS*    =S.TI       INSERT CREATED LEVEL
            JMP     LVL1        DISPLAY STORAGE EXCEEDED
            LAC     LVL5        GET POINTER TO CREATED LEVEL
            JMP     LVL3+2      RETURN
LVL1        LAC     LVL5        GET POINTER TO LEVEL
            JMS     ATTR        GET FIRST ATTRIBUTE
            $DC     0           LVL PROGRAMMING ERROR
            LMQ                 SET UP PARAMETER
            LAC     LVL5        GET POINTER TO LEVEL
            JMS*    =S.TR       REMOVE ATTRIBUTE
            $DC     0           LVL PROGRAMMING ERROR
            LAC     LVL5        GET POINTER TO LEVEL
            JMS*    =S.TD       DESTROY LEVEL
            $DC     0           LVL PROGRAMMING ERROR
            JMP     LVL3+3      RETURN
LVL2        LAC     LVL5        GET POINTER TO LEVEL
            JMS*    =S.TD       DESTROY LEVEL
            $DC     0           LVL PROGRAMMING ERROR
LVL3        ISZ     LVL+2       INCREMENT RETURN POINTER
            ISZ     LVL+2       INCREMENT RETURN POINTER
            ISZ     LVL+2       INCREMENT RETURN POINTER
            ISZ     LVL+2       INCREMENT RETURN POINTER
            JMS*    =T.U        UNLOCK LVL & RETURN
            $DC     LVL
*CALLING SEQUENCE:
*           JMS     BUTN
*           $DC     ----        (LOC CONTAINING POINTER TO OWNER)
*           $DC     ----        (Y COORDINATE)
*           $DC     ----        (X COORDINATE)
*           $DC     ----        (DISPLAY PARAMETER)
*           $DC     ----        (SERVICE TASK ADDRESS)
*           ----                (RETURN IF DISPLAY STORAGE EXCEEDED)
*           ----                (RETURN IF SUCCESSFUL)
*AC CONTENT  ON ENTRY:
```

```
*          POINTER TO STRUCTURE FOR BUTTON DISPLAY
*AC CONTENT  ON RETURN:
*          POINTER TO LIGHT BUTTON LEVEL
BUTN       $DC     0
           JMS*    =T.L        SET REENTRY LOCK
           $DC     0
           DAC     BUTN3       SAVE POINTER TO STRUCTURE
           LAW     -4          GET LVL PARAMETER COUNT
           DAC     BUTN4       INITIALIZE COUNTER
           LAC     =BUTN1      GET ADDRESS OF FIRST LVL PARAMETER
           DAC     BUTN5       INITIALIZE POINTER
           LAC*    BUTN+2      GET BUTN PARAMETER
           DAC*    BUTN5       STORE AS LVL PARAMETER
           ISZ     BUTN+2      INCREMENT BUTN PARAMETER POINTER
           ISZ     BUTN5       INCREMENT LVL PARAMETER POINTER
           ISZ     BUTN4       INCREMENT COUNTER & SKIP IF DONE
           JMP     *-5         COPY NEXT PARAMETER
           LAC     BUTN3       GET POINTER TO STRUCTURE
           JMS     LVL         GENERATE INTERMEDIATE LEVEL
BUTN1      $DC     0           LOC CONTAINING POINTER TO OWNER
           $DC     0           Y COORDINATE
           $DC     0           X COORDINATE
           $DC     0           DISPLAY PARAMETER
           JMP     BUTN2       DISPLAY STORAGE EXCEEDED
           DAC     BUTN3       SAVE POINTER TO LEVEL
           LAC*    BUTN+2      GET ADDRESS OF SERVICE TASK
           LMQ                 SET UP PARAMETER
           LAC     BUTN3       GET POINTER TO LEVEL
           JMS*    =S.LL       SENSITIZE LEVEL TO LPSI
           LAC     BUTN3       GET POINTER TO LEVEL
           ISZ     BUTN+2      INCREMENT RETURN POINTER
BUTN2      ISZ     BUTN+2      INCREMENT RETURN POINTER
           JMS*    =T.U        UNLOCK BUTN & RETURN
           $DC     BUTN
```

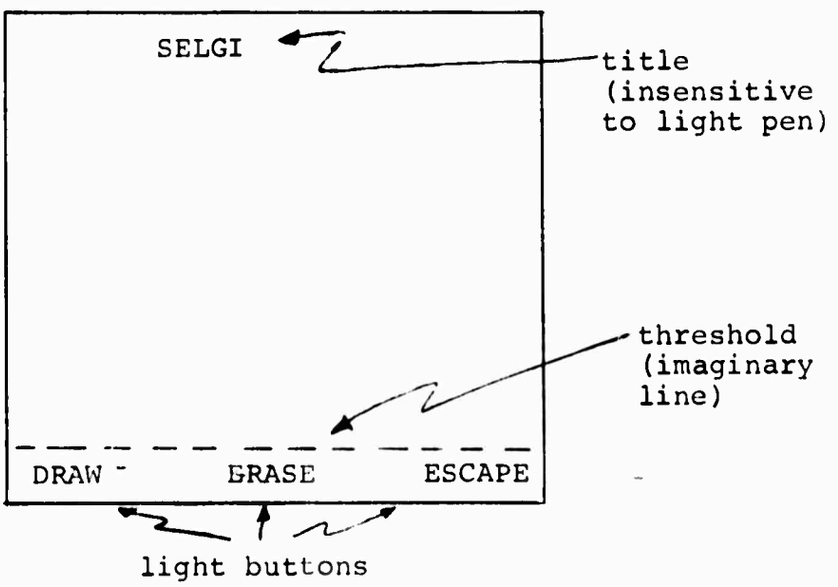
```
*CALLING SEQUENCE:
*          JMS     BUTX
*          $DC     ----       (ADDRESS OF TEXT LIST)
*          $DC     ----       (LOC CONTAINING POINTER TO OWNER)
*          $DC     ----       (Y COORDINATE)
*          $DC     ----       (X COORDINATE)
*          $DC     ----       (DISPLAY PARAMETER)
*          $DC     ----       (SERVICE TASK ADDRESS)
*          ----               (RETURN IF DISPLAY STORAGE EXCEEDED)
*          ----               (RETURN IF SUCCESSFUL)
*AC CONTENT  ON RETURN:
*          POINTER TO LIGHT BUTTON LEVEL
BUTX       $DC     0
           JMS*    =T.L       SET REENTRY LOCK
           $DC     0
           LAC*    BUTX+2     GET ADDRESS OF TEXT LIST
           JMS*    =L.D       CREATE TEXT LEAF
           JMP     BUTX4      DISPLAY STORAGE EXCEEDED
           DAC     BUTX7      SAVE POINTER TEXT LEAF
           LAW     -6         LOAD AC WITH -6
           DAC     BUTX5      SET PARAMETER COUNTER
           LAC     =BUTX2     GET ADDRESS OF BUTN CALL
           DAC     BUTX6      SET PARAMETER POINTER
BUTX1      ISZ     BUTX+2     ADVANCE TO NEXT PARAMETER
           ISZ     BUTX6      INCREMENT PARAMETER POINTER
           ISZ     BUTX5      SKIP IF NOT PARAMETER
           SKP                MOVE PARAMETER
           JMP     BUTX2-1    CALL BUTN
           LAC*    BUTX+2     GET PARAMETER
           DAC*    BUTX6      STORE PARAMETER
           JMP     BUTX1      MOVE NEXT PARAMETER
           LAC     BUTX7      GET POINTER TO TEXT LEAF
BUTX2      JMS     BUTN       CREATE LIGHT BUTTON
           $DC     0          LOC CONTAINING POINTER TO OWNER
           $DC     0          Y COORDINATE
```

```
          $DC     0          X COORDINATE
          $DC     0          DISPLAY PARAMETER
          $DC     0          SERVICE TASK ADDRESS
          JMP     BUTX3+2    DISPLAY STORAGE EXCEEDED
          ISZ     BUTX+2     INDICATE SUCCESS
BUTX3     JMS*    =T.U       UNLOCK BUTX & RETURN
          $DC     BUTX
          LAC     BUTX7      GET POINTER TO TEXT LEAF
          JMS*    =S.LL      DESTROY TEXT LEAF
          JMP     BUTX3      RETURN
BUTX4     LAC     BUTX+2     GET RETURN POINTER
          TAD     =6         ADVANCE PAST PARAMETER LIST
          DAC     BUTX+2     SET FAILURE RETURN POINTER
          JMP     BUTX3      RETURN
*CALLING SEQUENCE:
*         JMS CHEW
*         ----               (RETURN)
*AC CONTENT  ON ENTRY:
*         POINTER TO MAXIMUM ELEMENT IN THE STRUCTURE
*TO BE CHEWED
*THE MAXIMUM ELEMENT SPECIFIED MUST OWN ALL LEVELS
*WHICH OWN ELEMENTS OF THE STRUCTURE.
CHEW      $DC     0
          JMS*    =T.L       SET REENTRY LOCK
          $DC     0
          DAC     CHEW6      SAVE POINTER TO STRUCTURE
          LAC     =CHEWQ     GET ADDRESS OF WORD QUEUE
          JMS*    =Q.C       CLEAR WORD QUEUE
CHEW1     LAC*    CHEW6      GET FIRST WORD FROM STRUCTURE
          SNA                SKIP IF ITEM NOT ALREADY DELETED
          JMP     CHEW5      GET NEXT ITEM FROM QUEUE
          SAD     =762010    SKIP IF NOT TEXT LEAF
          JMP     CHEW4      DESTROY TEXT LEAF
          LAC     CHEW6      GET POINTER TO STRUCTURE
          AND     =70000     GET BREAK FIELD BITS
          SAD     =10000     SKIP IF NOT LEVEL
```

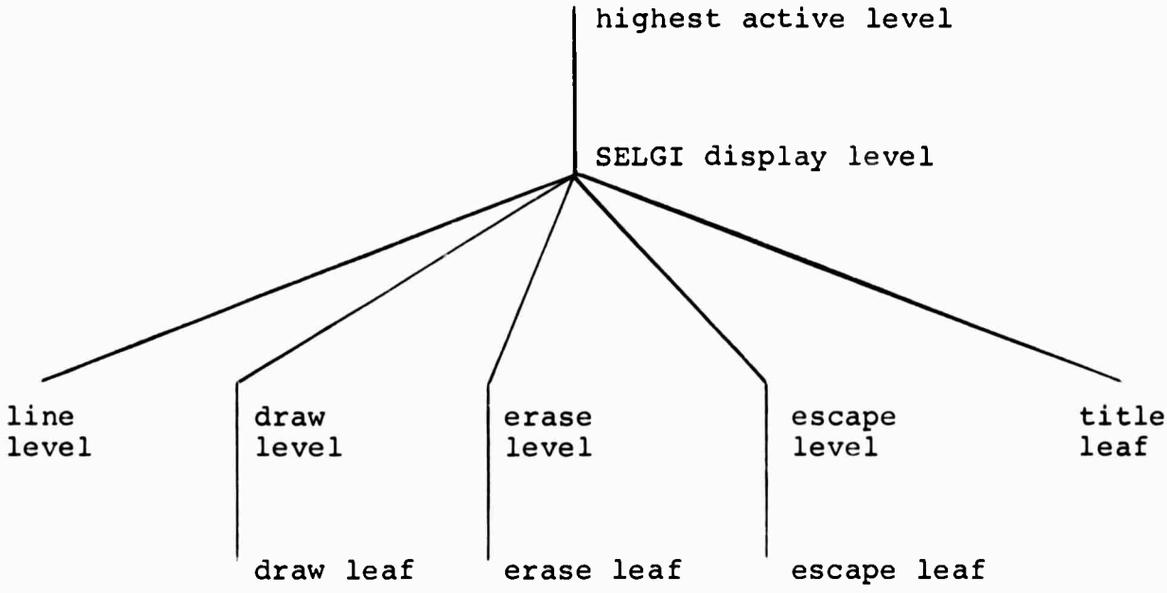
```
            SKP                 DESTROY LEVEL
            JMP     CHEW5       GET NEXT ITEM FROM QUEUE
CHEW2       LAC     CHEW6       GET POINTER TO LEVEL
            JMS     ATTR        GET FIRST ATTRIBUTE FROM LEVEL
            JMP     CHEW3       LEVEL IS EMPTY
            DAC     CHEW7       SAVE POINTER TO ATTRIBUTE
            LMQ                 SET UP PARAMETER
            LAC     CHEW6       GET POINTER TO LEVEL
            JMS*    =S.TR       REMOVE ATTRIBUTE
            $DC     0           CHEW PROGRAMMING ERROR
            LAC     CHEW7       GET POINTER TO ATTRIBUTE
            LMQ                 SET UP PARAMETER
            LAC     =CHEWQ      GET ADDRESS OF WORD QUEUE
            JMS*    =Q.A        ADD ATTRIBUTE TO QUEUE
            $DC     0           WORD QUEUE NOT LARGE ENOUGH
            JMP     CHEW2       PUT NEXT ATTRIBUTE IN QUEUE
CHEW3       LAC     CHEW6       GET POINTER TO LEVEL
            JMS*    =S.TD       DESTROY LEVEL
            $DC     0           CHEW PROGRAMMING ERROR
            JMP     CHEW5       CHEW UP NEXT ITEM
CHEW4       LAC     CHEW6       GET POINTER TO TEXT LEAF
            JMS*    =L.L        DESTROY TEXT LEAF
CHEW5       LAC     =CHEWQ      GET ADDRESS OF WORD QUEUE
            JMS*    =Q.F        GET NEXT ITEM FROM QUEUE
            JMP     *+3         QUEUE EMPTY
            DAC     CHEW6       SAVE POINTER TO ITEM
            JMP     CHEW1       CHEW UP ITEM FROM QUEUE
            JMS*    =T.U        UNLOCK CHEW & RETURN
            $DC     CHEW
CHEWQ       $DC     *+200
            $DC      200
*CALLING SEQUENCE:
*           JMS     ATTR
*           ----                (RETURN IF NO MORE ATTRIBUTES)
*           ----                (RETURN IF ATTRIBUTE FOUND)
```

```
*AC CONTENTS ON ENTRY:
*          ADDRESS OF LEVEL
ATTR       $DC     0
           TAD     =7          FORM POINTER TO POINTER TO NODE
           DAC     ATTR2       SAVE POINTER TO POINTER TO NODE
           LAC*    ATTR2       GET POINTER TO NODE (OR TAIL)
           DAC     ATTR2       SAVE POINTER TO NODE (OR TAIL)
           LAC*    ATTR2       GET FIRST WORD FROM NODE (OR TAIL)
           AND     =777770     TRUNCATE BREAK FIELD
           SAD     =762010     SKIP IF NOT NODE
           SKP                 NODE FOUND
           JMP*    ATTR        NO MORE ATTRIBUTES
           ISZ     ATTR2       FORM POINTER TO SECOND LOC IN NODE
           LAC*    ATTR2       GET POINTER TO ATTRIBUTE
           ISZ     ATTR        INDICATE SUCCESS
           JMP*    ATTR        RETURN
```

As an example of how these subroutines might be used, consider a task called SELGI which allows the operator to draw unrelated straight lines on the display with the light pen. More specifically, when the task is begun, it allocates the display and displays the following:



The elements of this display are arranged in the following structure:



The SELGI display level is set to scale x2, each light button level is sensitized to the light pen sense indicator, and the line level (into which all lines drawn by the operator will be inserted) has coordinates at the center of the screen.

When the light pen is pointed at the DRAW light button, the task DRAW is scheduled. The task DRAW then starts tracking on the DRAW light button, and waits (through the use of T.P) until the operator loses tracking. Then, if the Y tracking coordinate is above the threshold line, a line of length one point (which appears as a point on the display) is inserted into the line level such that it appears at the coordinates where tracking was lost. Otherwise, no line is generated. (The DRAW light button blinks while tracking is in process for this operation.) Up to 64 lines may be created in this manner.

If the light pen is now pointed at any of the unit-length lines (points) on the screen, tracking is started, and one end of the line is affixed to the tracking cross. The line

then may be stretched by moving the affixed end point to some other position on the screen. If the light pen is now pointed at any line which is longer than one point, tracking is started, and the end point of the line which is closer to the tracking cross is affixed to the tracking cross and may be moved to any position on the screen.

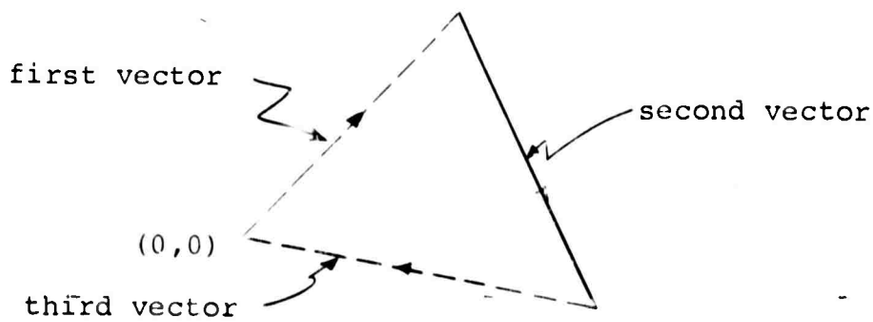
If the light pen is pointed at the ERASE light button, this light button starts blinking. If, while the ERASE light button is blinking, the light pen is pointed at some line on the screen, the line is removed from the line level, the storage which it occupied is salvaged, and the blinking of the ERASE light button is stopped.

If the light pen is pointed at the ESCAPE light button, the entire display structure created by SELGI is destroyed via the subroutine CHEW. The task SELGI then releases the display and terminates.

Lines are represented internally in this program by leaves which have the following format:

```
VEC     ENTER VECTOR MODE
----    Y DISPLACEMENT (NONINTENSIFIED)
----    X DISPLACEMENT (NO ESCAPE)
----    Y DISPLACEMENT (INTENSIFIED)
----    X DISPLACEMENT (NO ESCAPE)
----    Y DISPLACEMENT (NONINTENSIFIED)
----    X DISPLACEMENT (ESCAPE)
POP     END OF LEAF
```

Each leaf actually represents a triangle with two nonintensified sides. This scheme permits the end points of the line to occur anywhere on the screen:



```
SELGI   LAW    1          GET DISPLAY ALLOCATION MASK
        JMS*   =T.A       ALLOCATE DISPLAY
        LAC    =LINES     GET ADDRESS OF LINE STORAGE AREA
        DAC    DIS        SET STORAGE POINTER
        LAW    -1000      LOAD AC WITH -512
        DAC    FRM        SET STORAGE COUNTER
        DZM*   DIS        CLEAR STORAGE LOCATION
        ISZ    DIS        INCREMENT STORAGE POINTER
        ISZ    FRM        SKIP IF STORAGE AREA CLEARED
        JMP    *-3        CLEAR NEXT STORAGE LOCATION
        JMS*   =S.LH      GET ADDRESS OF HIGHEST ACTIVE LEVEL
        DAC    HAL        SAVE ADDRESS OF HIGHEST ACTIVE LEVEL
        LAC    =TXT       GET ADDRESS OF TITLE TEXT LIST
        JMS*   =L.D       CREATE TEXT LEAF
        JMP    END        DISPLAY STORAGE EXCEEDED
        DAC    DIS        SAVE POINTER TO TITLE LEAF
        JMS    LVL        GENERATE SELGI DISPLAY LEVEL
        $DC    HAL        POINTER TO HIGHEST ACTIVE LEVEL
        $DC    360        Y COORDINATE
        $DC    -34        X COORDINATE
        $DC    500        SCALE X2
        JMP    FAIL       DISPLAY STORAGE EXCEEDED
        DAC    FRM        SAVE POINTER TO SELGI DISPLAY LEVEL
        JMS    BUTX       GENERATE DRAW LIGHT BUTTON
        $DC    TXT1       DRAW TEXT LIST
        $DC    FRM        POINTER TO SELGI DISPLAY LEVEL
        $DC    -750       Y COORDINATE
        $DC    -344       X COORDINATE
        $DC    0          NULL DISPLAY PARAMETER
        $DC    DRAW       DRAW SERVICE TASK
        JMP    END        DISPLAY STORAGE EXCEEDED
        JMS    BUTX       GENERATE ERASE LIGHT BUTTON
        $DC    TXT2       ERASE TEXT LIST
        $DC    FRM        POINTER TO SELGI DISPLAY LEVEL
        $DC    -750       Y COORDINATE
```

```
$DC     10          X COORDINATE
$DC     0           NULL DISPLAY PARAMETER
$DC     ERASE       ERASE SERVICE TASK
JMP     END         DISPLAY STORAGE EXCEEDED
JMS     BUTX        GENERATE ESCAPE LIGHT BUTTON
$DC     TXT3        ESCAPE TEXT LIST
$DC     FRM         POINTER TO SELGI DISPLAY LEVEL
$DC     -750        Y COORDINATE
$DC     354         X COORDINATE
$DC     0           NULL DISPLAY PARAMETER
$DC     ESCAPE      ESCAPE SERVICE TASK
JMP     END         DISPLAY STORAGE EXCEEDED
JMS*    =S.TL       CREATE LINE LEVEL
JMP     END         DISPLAY STORAGE EXCEEDED
DAC     DIS         SAVE POINTER TO LINE LEVEL
LMQ                 SET UP PARAMETER
LAC     FRM         GET POINTER TO SELGI DISPLAY LEVEL
JMS*    =S.TI       INSERT LINE LEVEL
JMP     FAIL        DISPLAY STORAGE EXCEEDED
LAW     60          GET LIGHT PEN ON PARAMETER
LMQ                 SET UP PARAMETER
LAC     DIS         GET POINTER TO LINE LEVEL
JMS*    =S.LP       ENABLE LIGHT PEN ON LINE LEVEL
LAW     -360        GET Y COORDINATE
LMQ                 SET UP PARAMETER
LAC     DIS         GET POINTER TO LINE LEVEL
JMS*    =S.LY       SET Y COORDINATE OF LINE LEVEL
LAC     -34         GET X COORDINATE
LMQ                 SET UP PARAMETER
LAC     DIS         GET POINTER TO LINE LEVEL
JMS*    =S.LX       SET X COORDINATE OF LINE LEVEL
LAC     =MOVE       GET ADDRESS OF LINE MOVING TASK
JMS*    =D.P        SET LIGHT PEN FLAG SERVICE
JMS*    =D.E        ENABLE DISPLAY INTERRUPTS
DZM     ESCAPE+1    CLEAR ESCAPE SWITCH
```

```
        LAC     ESCAPE+1    GET ESCAPE SWITCH
        SZA                 SKIP IF ESCAPE NOT PENDING
        JMP     END         TERMINATE SELGI
        SKP                 PREPARE TO SCHEDULE NEXT LOCATION
        JMP     *-4         CHECK ESCAPE SWITCH
        JMS*    =T.P        SCHEDULE PREVIOUS LOCATION
FAIL    LAC     DIS         GET POINTER TO NONACTIVE STRUCTURE
        JMS     CHEW        DESTROY NONACTIVE STRUCTURE
END     LAC     HAL         GET POINTER TO HIGHEST ACTIVE LEVEL
        JMS     CHEW        DESTROY ACTIVE STRUCTURE
        CLA                 GET NULL LIGHT PEN FLAG SERVICE
        JMS*    =D.P        SET NULL LIGHT PEN SERVICE
        LAW     1           GET DISPLAY ALLOCATION MASK
        JMS*    =T.R        RELEASE DISPLAY
        JMA*    =T.F        TERMINATE
DRAW    LAW     -720        GET INITIAL X TRACKING COORDINATE
        LMQ                 SET UP PARAMETER
        LAW     -730        GET INITIAL Y TRACKING COORDINATE
        JMS*    =X.I        INITIALIZE TRACKING
        CLA                 PREPARE TO READ OWNER 0 LEVELS BACK
        JMS*    =D.O        READ ADDRESS OF DRAW LEVEL
        $DC     0           PROGRAMMING ERROR IF D.O FAILS
        JMS*    =S.LBE      ENABLE BLINK ON DRAW LIGHT BUTTON
        JMS*    =X.S        SKIP IF TRACKING HAS BEEN LOST
        JMS*    =T.P        CHECK TRACKING AGAIN
        JMS*    =X.Y        READ Y TRACKING COORDINATE
        TAD     =700        FORM THRESHOLD CHECK
        SPA                 SKIP IF LINE IS TO BE CREATED
        JMP     DRAW1       IGNORE ATTEMPT TO CREATE LINE
        LAC     =LINES      GET POINTER TO LINE STORAGE
        DAC     FRM         SET STORAGE POINTER
        LAW     -100        GET MAXIMUM LINE COUNT
        DAC     CNT         SET LINE COUNTER
        LAC*    FRM         GET FIRST WORD OF LINE BLOCK
        SNA                 SKIP IF LINE BLOCK IN USE
```

```
        JMP     *+7         LINE BLOCK IS AVAILABLE
        LAC     FRM         GET STORAGE POINTER
        TAD     =10         FORM ADDRESS OF NEXT LINE BLOCK
        DAC     FRM         SET STORAGE POINTER TO NEXT BLOCK
        ISZ     CNT         SKIP IF NO MORE LINE STORAGE
        JMP     *-7         CHECK AVAILABILITY OF LINE BLOCK
        JMP     DRAW1       IGNORE ATTEMPT TO CREATE LINE
        LAW     1121        GET VEC INSTRUCTION
        DAC*    FRM         STORE IN FIRST LOCATION OF LINE BLOCK
        LAC     FRM         GET POINTER TO LINE BLOCK
        TAD     =7          FORM POINTER TO LAST WORD IN BLOCK
        DAC     CNT         SAVE POINTER TO LAST WORD IN BLOCK
        LAW     3000        GET POP INSTRUCTION
        DAC*    CNT         STORE IN LAST WORD IN BLOCK
        LAC     FRM         GET POINTER TO LINE BLOCK
        JMS     FIXBGN      SET 1ST END POINT TO TRACKING COORD
        LAC     FRM         GET POINTER TO LINE BLOCK
        JMS     FIXEND      SET 2ND END POINT TO TRACKING COORD
        LAC     FRM         GET POINTER TO LINE BLOCK
        LMQ                 SET UP PARAMETER
        LAC     DIS         GET POINTER TO LINE LEVEL
        JMS*    =S.TI       INSERT LINE BLOCK
        NOP                 DISPLAY STORAGE EXCEEDED
DRAW1   CLA                 PREPARE TO READ OWNER 0 LEVELS BACK
        JMS*    =D.O        READ ADDRESS OF DRAW LEVEL
        $DC     0           PROGRAMMING ERROR IF D.O FAILS
        JMS*    =S.LBD      STOP BLINK OF DRAW LIGHT BUTTON
        JMS*    =D.E        ENABLE DISPLAY INTERRUPTS
        JMS*    =T.F        TERMINATE
MOVE    JMS*    =D.Y        READ Y DISPLAY COORDINATE
        DAC     MOVEY       SAVE Y DISPLAY COORDINATE
        JMS*    =D.X        READ X DISPLAY COORDINATE
        LMQ                 SET UP PARAMETER
        LAC     MOVEY       GET Y DISPLAY COORDINATE
        JMS*    =X.I        INITIALIZE TRACKING
```

```
        CLA             PREPARE TO READ OWNER 0 LEVELS BACK
        JMS*    =D.O    READ ADDRESS OF LINE LEAF
        $DC     0       PROGRAMMING ERROR IF D.O FAILS
        DAC     MOVE1   SAVE POINTER TO LINE LEAF
        DAC     MOVE2   SAVE COPY OF POINTER TO LINE LEAF
        ISZ     MOVE2   FORM POINTER TO FIRST Y DISPLACEMENT
        TAD     =5      FORM POINTER TO THIRD Y DISPLACEMENT
        DAC     MOVE3   SAVE POINTER TO THIRD Y DISPLACEMENT
        LAC*    MOVE2   GET FIRST Y DISPLACEMENT
        XOR     =2000   INVERT SIGN BIT
        JMS*    =C.CB   CONVERT TO TWO'S COMPLEMENT
        LLSS    1       MULTIPLY BY 2
        TAD     MOVEY   ADD Y DISPLAY COORDINATE
        GSM             FORM ABSOLUTE VALUE
        DAC     MOVE4   SAVE FOR LATER COMPARISON
        LAC*    MOVE3   GET THIRD Y DISPLACEMENT
        JMS*    =C.CB   CONVERT TC TWO'S COMPLEMENT
        LLSS    1       MULTIPLY BY 2
        TAD     MOVEY   ADD Y DISPLAY COORDINATE
        GSM             FORM ABSOLUTE VALUE
        CMA             FORM NEGATIVE OF ABSOLUTE VALUE
        TAD     MOVE4   ADD DISPLACEMENT FROM OTHER END
        SMA             SKIP IF CLOSER TO FIRST Y DISPLACEMENT
        JMP     *+3     CLOSER TO SECOND Y DISPLACEMENT
        JMS     WATCH   ENTER UPDATING TASK
        JMS     FIXBGN  PARAMETER FOR UPDATING TASK
        JMS     WATCH   ENTER UPDATING TASK
        JMS     FIXEND  PARAMETER FOR UPDATING TASK
WATCH   $DC     0
        LAC     MOVE1   GET POINTER TO LINE LEAF
        XCT*    WATCH   UPDATE AFFIXED END POINT
        JMS*    =X.S    SKIP IF TRACKING NOT IN PROCESS
        JMP     *+6     SCHEDULE NEXT UPDATING
        LAW     -40     LOAD AC WITH -32
        JMS*    =N.C    WAIT ABOUT HALF A SECOND
```

```
        JMS*    =D.E        ENABLE DISPLAY INTERRUPTS
        JMS*    =T.F        TERMINATE
        JMP     WATCH+1     UPDATE END  POINT
        JMS*    =T.P        SCHEDULE PREVIOUS LOCATION
ERASE   LAC     =DELETE     GET ADDRESS OF LINE DELETE TASK
        JMS*    =D.P        SET LIGHT PEN FLAG SERVICE
        CLA                 PREPARE TO READ OWNER 0 LEVELS BACK
        JMS*    =D.O        GET POINTER TO ERASE LEVEL
        $DC     0           PROGRAMMING ERROR IF D.O FAILS
        DAC     ERS         SAVE POINTER TO ERASE LEVEL
        JMS*    =S.LBE      START BLINKING ERASE LIGHT BUTTON
        JMS*    =D.E        ENABLE DISPLAY INTERRUPTS
        JMS*    =T.F        TERMINATE
DELETE  LAC     ERS         GET POINTER TO ERASE LEVEL
        JMS*    =S.LBD      STOP BLINKING ERASE LIGHT BUTTON
        CLA                 PREPARE TO READ OWNER 0 LEVELS BACK
        JMS*    =D.O        GET POINTER TO LINE LEAF
        $DC     0           PROGRAMMING ERROR IF D.O FAILS
        DAC     FRM         SAVE POINTER TO LINE LEAF
        LMQ                 SET UP PARAMETER
        LAC     DIS         GET POINTER TO LINE LEVEL
        JMS*    =S.TR       REMOVE LINE LEAF
        $DC     0           PROGRAMMING ERROR IF S.TR FAILS
        DZM*    FRM         DESTROY LINE LEAF
        LAC     =MOVE       GET ADDRESS OF LINE MOVING TASK
        JMS*    =D.P        SET LIGHT PEN SERVICE
        LAW     -40         LOAD AC WITH -32
        JMS*    =N.C        WAIT ABOUT HALF A SECOND
        JMS*    =D.E        ENABLE DISPLAY INTERRUPTS
        JMS*    =T.F        TERMINATE
ESCAPE  JMS     *+1         SET ESCAPE SWITCH
        $DC     0           ESCAPE SWITCH
        JMS*    =T.F        TERMINATE
FIXBGN  $DC     0
        JMS     FIXRD       SET UP POINTERS FOR FIXING LEAF
```

```
            LAC     FIXY      GET Y TRACKING COORDINATE
            DAC*    FIX1      SET FIRST Y DISPLACEMENT
            LAC     FIXX      GET X TRACKING COORDINATE
            DAC*    FIX2      SET FIRST X DISPLACEMENT
            JMS     FIXFIX    CORRECT INTENSIFIED VECTOR
            JMP*    FIXBGN    RETURN
FIXEND      $DC     0
            JMS     FIXRD     SET UP POINTERS FOR FIXING LEAF
            LAC     FIXY      GET Y TRACKING COORDINATE
            XOR     =2000     INVERT SIGN BIT
            DAC*    FIX5      SET THIRD Y DISPLACEMENT
            LAC     FIXX      GET X TRACKING COORDINATE
            XOR     =6000     INVERT SIGN BIT, SET ESCAPE BIT
            DAC*    FIX6      SET THIRD X DISPLACEMENT
            JMS     FIXFIX    CORRECT INTENSIFIED VECTOR
            JMP*    FIXEND    RETURN
FIXRD       $DC     0
            TAD     =1        FORM POINTER TO FIRST Y DISPLACEMENT
            DAC     FIX1      SAVE
            TAD     =1        FORM POINTER TO FIRST X DISPLACEMENT
            DAC     FIX2      SAVE
            TAD     =1        FORM POINTER TO SECOND Y DISPLACEMENT
            DAC     FIX3      SAVE
            TAD     =1        FORM POINTER TO SECOND X DISPLACEMENT
            DAC     FIX4      SAVE
            TAD     =1        FORM POINTER TO THIRD Y DISPLACEMENT
            DAC     FIX5      SAVE
            TAD     =1        FORM POINTER TO THIRD X DISPLACEMENT
            DAC     FIX6      SAVE
            JMS*    =X.Y      READ Y TRACKING COORDINATE
            LRSS    1         DIVIDE BY 2
            JMS*    =C.BC     CONVERT TO DISPLAY COORDINATE
            DAC     FIXY      SAVE
            JMS*    =X.X      READ X TRACKING COORDINATE
            LRSS    1         DIVIDE BY 2
```

```
            JMS*    =C.BC     CONVERT TO DISPLAY COORDINATE
            DAC     FIXX      SAVE
            JMP*    FIXRD     RETURN
FIXFIX      $DC     0
            LAC*    FIX1      GET FIRST Y DISPLACEMENT
            JMS*    =C.CB     CONVERT TO TWOS COMPLEMENT
            DAC     FIXY      SAVE
            LAC*    FIX5      GET THIRD Y DISPLACEMENT
            JMS*    =C.CB     CONVERT TO TWOS COMPLEMENT
            TAD     FIXY      ADD FIRST Y DISPLACEMENT
            JMS*    =C.BC     CONVERT TO DISPLAY COORDINATE
            SZA               SKIP IF Y DISPLACEMENTS WERE EQUAL
            JMP     *+7       CONVERTED VALUE IS NONZERO
            LAC*    FIX5      GET THIRD Y DISPLACEMENT
            JMS*    =C.CB     CONVERT TO TWOS COMPLEMENT
            TAD     =1        MAKE DIFFERENT FROM 1ST Y DISPLACEMENT
            JMS*    =C.BC     CONVERT TO DISPLAY COORDINATE
            DAC*    FIX5      STORE MODIFIED THIRD Y DISPLACEMENT
            LAW     1         GET DISPLACEMENT OF 1
            XOR     =6000     SET ESCAPE BIT, INVERT SIGN BIT
            DAC*    FIX3      STORE SECOND Y DISPLACEMENT
            LAC*    FIX2      GET FIRST X DISPLACEMENT
            JMS*    =C.CB     CONVERT TO TWOS COMPLEMENT
            DAC     FIXX      SAVE
            LAC*    FIX6      GET THIRD X DISPLACEMENT
            JMS*    =C.CB     CONVERT TO TWOS COMPLEMENT
            TAD     FIXX      ADD FIRST X DISPLACEMENT
            JMS*    =C.BC     CONVERT TO DISPLAY COORDINATE
            XOR     =2000     INVERT SIGN BIT
            DAC*    FIX4      SET SECOND X DISPLACEMENT
            JMP*    FIXFIX    RETURN
TXT         $DC     2
            $TEXT   "SELGI"
TXT1        $DC     2
            $TEXT   "DRAW"
```

```
TXT2        $DC     2
            $TEXT   "ERASE"
TXT3        $DC     2
            $TEXT   "ESCAPE"
LINES       $DS     1000
            $END
```

3.11 Text List Manipulation

A structure which may be used to represent efficiently strings of text in core is called a "text list." A text list consists of a word which contains a number m which represents the length of the list, followed by m words, each of which contains three 6-bit characters. As an example, a text list which represents the string

A SIMPLE EXAMPLE

is the following (in octal form):

000006
127634
222631
251676
164112
263125
167777

This text list may easily be represented in assembly language via the TEXT pseudo-op:

```
LIST        $DC     6
            $TEXT   "A SIMPLE EXAMPLE"
```

The address of the text list is the address of its first word. In this example, LIST is a symbol whose value is the address of the text list.

A "text leaf" is a representation of a text list as a display leaf. The leaf is composed of a series of push jumps to various character generation subroutines within the System.

A carriage return, however, is represented explicitly in the text leaf by three words which generate a vector which restores the X coordinate to its value just before the display control enters the text leaf. An additional vector is included at the end of the text leaf to restore both the X and Y coordinates. The high-order six bits of the second word of each push jump contain the 6-bit code for the character which the push jump represents. Each character is drawn in increment mode and is 7 points high by 5 points wide. The trailing space, which is produced by each character generation subroutine, is 3 points wide.

As an example of a text leaf, consider the following text list:

```
LEAF      $DC     10
          $TEXT   "EXAMPLE OF"
          $DC     747577
          $TEXT   "2 LINES"
```

The text leaf which would be produced from this text list is the following:

```
762010
16----
762010
41----
762010
12----
762010
26----
762010
31----
762010
25----
762010
16----
762010
76----
762010
30----
762010
17----
761121
400000
006120
```

```
762010
75----
762010
02----
762010
76----
762010
25----
762010
22----
762010
27----
762010
16----
762010
34----
761121
400020
006070
763000
```

The following system subroutines have been defined for manipulating text lists and text leaves:

L.T - The text list whose address is given in bits 3-17 of the AC is typed on the teletype.

L.D*- A text leaf is generated from the text list whose address is given in bits 3-17 of the AC. The address of the generated text leaf is returned in bits 3-17 of the AC. A failure return is made if the text leaf cannot be generated because of insufficient free display storage.

L.L - The text leaf whose address is given in bits 3-17 of the AC is destroyed, and the storage which it occupied is salvaged by the System.

4. IDLE-TIME TASK

The idle-time task, which is executed whenever the System is in system state (Section 2.3), interprets various keyboard commands which provide some functions which are useful for testing and modifying user tasks. These commands are described in Sections 4.1 through 4.5. Each command is given by typing only the underlined characters; the System will type all other characters shown.

4.1 Copy Functions

The command

FROM { TELETYPE / PAPER TAPE / CORE } TO { TELETYPE / PAPER TAPE / CORE / DISPLAY }

allows the operator to transfer data from teletype, paper tape, or core to teletype, paper tape, core, or the display. Many of these copy functions normally are specified by other names. For example, a copy from paper tape to core is called loading, a copy from core to teletype or from core to display is called a dump, a copy from teletype to core is called altering, etc.

When a transfer from teletype to any device other than core is specified, everything typed on the teletype up to the next character which maps into a 6-bit null character (Section 3.3) is transferred to the device specified. After a null character is typed, the idle-time task is ready for a new command. When copying from teletype to core, the following sequence of events occurs:

(1) The operator types a 5-digit octal address on the keyboard. If one character which he types is not an octal digit, it is interpreted as the first character of a new command, and the copy from teletype to core is terminated.

(2) The idle-time task types the content of the location specified on the current line of text.

(3) The operator types a 6-digit octal content to replace the content of the location specified on the current line of text. If he types a carriage return in place of one of the octal digits, the content of the location is left unchanged. If he types a character which is neither an octal digit nor a carriage return, the copy task proceeds with Step 1.

(4) The address of the location which immediately follows the one which was just examined (and perhaps modified) is typed. The copy task then proceeds with Step 2.

As an example of a copy from teletype to core, consider setting the content of location 23571_8 to 547521_8 and the content of location 23574_8 to 607213_8. This may be accomplished by either of the following procedures:

```
FROM TELETYPE TO CORE

23571     172356     547521
23572     543125                (carriage return)
23573     601241                (carriage return)
23574     760001     607213
23575     127123                (rubout)
FROM ----                       (new command)

FROM TELETYPE TO CORE

23571     172356     547521
23572     543125                (rubout)
23574     760001     607213
23575     127123                (rubout)
FROM ----                       (new command)
```

When a copy from paper tape to any device other than core is specified, the next alphanumeric record (Section 3.4.2) is read, and all binary records which are encountered before it

are ignored. (However, if the alphanumeric record is too long for the display, and a copy from paper tape to display is specified, only part of the alphanumeric record is read. The next part of the record may be displayed by another copy from paper tape to display.) Similarly, whenever a copy from paper tape to core is specified, the next binary record is read, and all alphanumeric records which are encountered before it are ignored.

When a copy from core to any device is specified, the specification of a block of core locations is requested from the operator. For example, the operator may dump locations 23571_8 through 23602_8 on the teletype as follows:

FROM CORE TO TELETYPE

BLOCK (23571, 23602)

23571 172356 543125 601241 760001 127123 127124 000200 000001
23601 000236 777777

A copy from core to core will also request the address of the first location in the block into which the information is to be moved. For example, locations 20052_8 through 20056_8 may be moved into locations 21521_8 through 21525_8 by the following command:

FROM CORE TO CORE

BLOCK (20052, 20056) TO 21521

Since the words in a block to be moved by a copy from core to core are moved one at a time, starting with the lowest address of the specified block, the following sequence of commands may be used to store zeros in all of core bank 1. (This is sometimes a useful operation to perform before loading a program to be debugged, since it stores illegal instructions througout core bank 1.)

FROM TELETYPE TO CORE

20000 172132 000000
20001 172312 (rubout)

FROM CORE TO CORE
BLOCK (20000, 37776) TO 20001

The copy from core to core in this example moves the zero from location 20000_8 into location 20001_8, then it moves the zero from location 20001_8 into location 20002_8, etc.

Copy functions to the display are constrained to a maximum of 64 characters per line and to 10 lines. For this reason, a maximum of 100_8 locations may be dumped on the screen at one time, and a copy from paper tape or teletype to display will be terminated at the end of 10 lines.

4.2 Scheduling of User Tasks

User tasks may be scheduled while in system state, but they will not be executed until user state is entered (Section 4.5). The command which accomplishes this is the following:

SCHEDULE _____

In the blanks after the word "schedule" the operator should type a 5-digit octal address where the task which he is scheduling begins. For example, a user task which starts at location 20571_8 may be scheduled by the following command:

SCHEDULE 20571

4.3 Clearing the Task Queue or Display Storage

The command

CLEAR { TASK QUEUE / DISPLAY STORAGE }

allows the operator to remove all user tasks scheduled by the command described in Section 4.2 from the task queue, or to clear the display storage area. When a copy function to the display is performed, the comment

NOT ENOUGH DISPLAY STORAGE

may be printed on the teletype, and the copy function will not be completed. The facility of clearing the display storage area is provided to allow the operator to destroy all display structures to provide display storage for copy functions to the display.

4.4 Teletype tc Dataphone Transmission

Since most messages to be sent over the 201A dataphone to a remote computer from the teletype are record-oriented, rather than character-oriented, and since ASCII codes are accepted as standards for this type of communication, a copy from the teletype to the dataphone is handled in a different manner from other copy functions. If the command "#" is typed, all succeeding characters typed on the keyboard, up to the first carriage return, are sent over the dataphone as one record of ASCII characters. (Of course, any response to such a record which does not begin with the 8-bit character 000_8 will be typed by the 201-to-teleprinter task.) However, a rubout will delete a partially typed line, and the character "←" will delete the previous character on the line, if it exists. This command is terminated when the line is terminated or deleted. A record consisting of an enquiry (used as an end-of-record character) may be sent from the teletype by striking the "WRU" key when the idle-time task is expecting a new command.

4.5 Entering User State

The command

RUN

causes all user tasks which have been scheduled by the command described in Section 4.2 to become eligible for execution, and the idle-time task to be terminated. This causes the System to enter user state (Section 2.3).

5. SYSTEM CAPABILITY

The System was designed primarily to support user tasks which provide communication between the operator and the 339 via network diagrams and between the 339 and a large timesharing system. As can be determined by examination of the display structure, the display support provided by the System is easily applied to almost any display-oriented task which is two-dimensional in nature (e.g., network diagrams, two-dimensional Sketchpad programs, line-oriented text editing, etc.). The System offers no support for tasks which involve three-dimensional projection in that: (1) floating point arithmetic (which is almost essential for this type of task) is not provided, and (2) the display structure has no provision for storing the extra information required for three-dimensional projection.

Because a timesharing system is not always available to support preparation and testing of remote terminal programs, the philosophy behind the design of the system was to consider the remote terminal as an independent unit which considers the large timesharing system to be an I/O device. This differs from the philosophy, which is commonly applied to the design of remote terminal software systems, that the large timesharing system must be available to support the remote terminal system whenever the remote system is operating.

BIBLIOGRAPHY

1. 339 Programmed Buffered Display, DEC-09-I6FA-D, Digital Equipment Corporation, Maynard, Massachusetts, May 1968.

2. Mills, David L., I/O Extensions to RAMP, Memorandum 11, Concomp Project, University of Michigan, Ann Arbor, October 1967.

3. Mills, David L., RAMP: A PDP-8 Multiprogramming System for Real-Time Device Control, Memorandum, Concomp Project, University of Michigan, Ann Arbor, May 1967.

4. Mills, David L., The Data Concentrator, Technical Report No. 8, Concomp Project, University of Michigan, Ann Arbor, May 1968.

5. PDP-9 User Handbook, F-95, Digital Equipment Corporation, Maynard, Massachusetts, January 1968.

6. Wood, David E., A 201A Data Communication Adapter for the PDP-8: Preliminary Engineering Design Report, Memorandum 15, Concomp Project, University of Michigan, Ann Arbor, February 1968.

APPENDIX A -- LISTING OF THE EXECUTIVE SYSTEM

```
        $ORG 17732
        $TITLE                  SEL EXECUTIVE SYSTEM LOADER
IOT     $OPDM 700000
HLT     $OPD 740040

        IOT 3302                CLEAR ALL FLAGS
        JMP SYSTEM              START SYSTEM
.1      JMS .4                  READ FIRST LINE OF 3-LINE BLOCK
        SNA                     SKIP IF NONBLANK TAPE
        JMP *-2                 BLANK TAPE -- TRY AGAIN
        DAC .5                  SAVE FIRST LINE IMAGE
        AND .7                  REMOVE DATA BITS
        SAD .8                  SKIP IF DATA LINE
        SKP                     ORIGIN LINE
        JMP .2                  DATA LINE
        JMS .3                  FINISH ORIGIN WORD
        DAC .6                  SET LOCATION COUNTER
        JMP .1                  READ NEXT BLOCK
.2      JMS .3                  FINISH DATA WORD
        DAC* .6                 LOAD DATA WORD
        ISZ .6                  INCREMENT LOCATION COUNTER
        JMP .1                  READ NEXT BLOCK

.3      $DC 0
        JMS .4                  READ SECOND LINE
        LRS 6                   SHIFT DATA BITS INTO MQ
        LAC .5                  LOAD AC WITH FIRST LINE IMAGE
        LLS 6                   SHIFT CONCATENATED IMAGE INTO AC
        DAC .5                  SAVE CONCATENATED FIRST TWO LINES
        JMS .4                  READ THIRD LINE
        LRS 6                   SHIFT DATA BITS INTO MQ
        LAC .5                  LOAD AC WITH CONCATENATED IMAGE
        LLS 6                   SHIFT COMPLETED WORD INTO AC
        JMP* .3                 RETURN

.4      $DC 0
        IOT 104                 SELECT READER
        IOT 101                 SKIP IF LINE READY
```

```
        JMP *-1                 WAIT FOR FLAG
        IOT 112                 OVERRIDDEN "JMP .1-2" WHEN LOADED
        JMP* .4                 RETURN

.5      SDC 0
.6      SDC 0
.7      SDC 300
.8      SDC 100

        JMP .1                  OVERRIDES BOOTSTRAP LOCATION 0
```

```
        $TITLE          CONTROL DISPATCHER

        $ORG 1
        JMP I           INTERRUPT TRAP

        $ORG 21
        JMP ET          ILLEGAL INSTRUCTION TRAP
        JMP ES          SYSTEM RESTART

        $ORG 100
Q.C     $DC 0
        JMP QC          CLEAR QUEUE
Q.A     $DC 0
        JMP QA          ADD WORD TO BOTTOM OF QUEUE (F)
Q.I     $DC 0
        JMP QI          INSERT WORD ON TOP OF QUEUE (F)
Q.F     $DC 0
        JMP QF          FETCH WORD FROM TOP OF QUEUE (F)
T.S     $DC 0
        JMP TS          SCHEDULE TASK
T.P     $DC 0
        JMP TP          SCHEDULE PREVIOUS LOC & TERMINATE
T.F     $DC 0
        JMP TF          TERMINATE CURRENT TASK
T.A     $DC 0
        JMP TA          ALLOCATE I/O DEVICES UNDER MASK
T.R     $DC 0
        JMP TR          RELEASE I/O DEVICES UNDER MASK
T.L     $DC 0
        JMP TL          LOCK REENTRABLE SUBROUTINE
T.U     $DC 0
        JMP TU          UNLOCK REENTRABLE SUBROUTINE
C.B6    $DC 0
        JMP CB6         CONVERT BINARY TO 6-BIT OCTAL
C.6A    $DC 0
        JMP C6A         CONVERT 6-BIT TO ASCII
C.A6    $DC 0
        JMP CA6         CONVERT ASCII TO 6-BIT
C.CB    $DC 0
        JMP CCB         CONVERT DISPLAY COORDINATE TO BINARY
```

```
C.BC    $DC 0
        JMP CBC     CONVERT BINARY TO DISPLAY COORDINATE
B.FI    $DC 0
        JMP BFI     GET IMAGE FROM 201 INPUT BUFFER (F)
B.FO    $DC 0
        JMP BFO     SEND IMAGE TO 201 OUTPUT BUFFER (F)
B.R     $DC 0
        JMP BR      GET IMAGE FROM READER BUFFER (F)
B.P     $DC 0
        JMP BP      SEND IMAGE TO PUNCH BUFFER (F)
B.K     $DC 0
        JMP BK      GET 6-BIT CHAR FROM KEYBOARD BUFFER
B.T     $DC 0
        JMP BT      SEND 3 PACKED 6-BIT CHARS TO TP BUF
N.A     $DC 0
        JMP NA      CONVERT ANALOG TO DIGITAL
N.C     $DC 0
        JMP NC      SET CLOCK INTERVAL & SERVICE TASK
N.D1    $DC 0
        JMP ND1     SELECT D/A CONVERTER #1
N.D2    $DC 0
        JMP ND2     SELECT D/A CONVERTER #2
N.D3    $DC 0
        JMP ND3     SELECT D/A CONVERTER #3
P.T     $DC 0
        JMP PT      SET PUSH BUTTON SERVICE TASK
P.E     $DC 0
        JMP PE      ENABLE MANUAL OPN OF PUSH BUTTONS
P.D     $DC 0
        JMP PD      DISABLE MANUAL OPN OF PUSH BUTTONS
P.R     $DC 0
        JMP PR      READ PUSH BUTTONS
P.S     $DC 0
        JMP PS      SET PUSH BUTTONS
D.E     $DC 0
        JMP DE      ENABLE DISPLAY INTERRUPTS
D.D     $DC 0
        JMP DD      DISABLE DISPLAY INTERRUPTS
D.P     $DC 0
        JMP DP      SET LIGHT PEN FLAG SERVICE TASK
```

```
D.A     $DC  0
        JMP  DA      READ DISPLAY ADR ON LAST INTERRUPT
D.Y     $DC  0
        JMP  DY      READ Y DPY COORD ON LAST INTERRUPT
D.X     $DC  0
        JMP  DX      READ X DPY COORD ON LAST INTERRUPT
D.O     $DC  0
        JMP  DO      READ OWNER ON LAST INTERRUPT (F)
X.I     $DC  0
        JMP  XI      INITIALIZE TRACKING AT GIVEN COORDS
X.R     $DC  0
        JMP  XR      RESUME TRACKING
X.T     $DC  0
        JMP  XT      TERMINATE TRACKING
X.S     $DC  0
        JMP  XS      SKIP IF TRACKING NOT IN PROCESS (F)
X.Y     $DC  0
        JMP  XY      READ Y TRACKING COORDINATE
X.X     $DC  0
        JMP  XX      READ X TRACKING COORDINATE
S.TL    $DC  0
        JMP  STL     CREATE A LEVEL (F)
S.TD    $DC  0
        JMP  STD     DESTROY A LEVEL (F)
S.TI    $DC  0
        JMP  STI     INSERT SUBSTRUCTURE INTO LEVEL (F)
S.TR    $DC  0
        JMP  STR     REMOVE SUBSTRUCTURE FROM LEVEL (F)
S.LH    $DC  0
        JMP  SLH     GET ADDRESS OF HIGHEST ACTIVE LEVEL
S.LY    $DC  0
        JMP  SLY     TRANSLATE LEVEL IN Y DIRECTION
S.LX    $DC  0
        JMP  SLX     TRANSLATE LEVEL IN X DIRECTION
S.LP    $DC  0
        JMP  SLP     SET LEVEL PARAMETERS
S.LBE   $DC  0
        JMP  SLBE    ENABLE BLINK ON LEVEL
S.LBD   $DC  0
        JMP  SLBD    DISABLE BLINK ON LEVEL
```

```
S. C    $DC 0
        JMP SLC         COUNT SCALE AND/OR INTENSITY
S.LU    $DC 0
        JMP SLU         INTERRUPT UNCONDITIONALLY ON LEVEL
S.LS    $DC 0
        JMP SLS         INTERRUPT ON LEVEL IF ON SCREEN
S.LL    $DC 0
        JMP SLL         INTERRUPT ON LEVEL IF LPSI SET
S.LN    $DC 0
        JMP SLN         DISABLE INTERRUPT ON LEVEL
L.T     $DC 0
        JMP LT          SEND TEXT LIST TO TP BUFFER
L.D     $DC 0
        JMP LD          GENERATE TEXT LEAF (F)
L.L     $DC 0
        JMP LL          DESTROY TEXT LEAF

PDP1    $DS 204

PDP2    $DS 204
```

```
        STITLE                  DISPLAY CHARACTER GENERATOR

D00     INCR
        $DC 1272
        $DC 6251
        $DC 6057
        $DC 7516
        $DC 1570
        $DC 5172
        $DC 3726
        $DC 0
        POP
D01     INCR
        $DC 5160
        $DC 1472
        $DC 7255
        $DC 3737
        $DC 0
        POP
D02     INCR
        $DC 5271
        $DC 5152
        $DC 5364
        $DC 5537
        $DC 2774
        $DC 5417
        $DC 3020
        $DC 0
        POP
D03     INCR
        $DC 1252
        $DC 5760
        $DC 5152
        $DC 5354
        $DC 1051
        $DC 5253
        $DC 6455
        $DC 3737
        $DC 1000
        POP
```

```
D04     INCR
        SDC 1110
        SDC 5072
        SDC 7275
        SDC 6010
        SDC 5037
        SDC 1600
        POP
D05     INCR
        SDC 1252
        SDC 5760
        SDC 5162
        SDC 5374
        SDC 6270
        SDC 5037
        SDC 3616
        SDC 0
        POP
D06     INCR
        SDC 1252
        SDC 5760
        SDC 5152
        SDC 5364
        SDC 5572
        SDC 5170
        SDC 3736
        SDC 1600
        POP
D07     INCR
        SDC 5271
        SDC 5162
        SDC 7454
        SDC 5637
        SDC 3710
        SDC 0
        POP
D10     INCR
        SDC 1252
        SDC 5760
        SDC 5152
```

```
        $DC 5364
        $DC 5512
        $DC 6251
        $DC 6057
        $DC 5637
        $DC 2600
        POP
D11     INCR
        $DC 5270
        $DC 5162
        $DC 7453
        $DC 5251
        $DC 6057
        $DC 5637
        $DC 2600
        POP
D12     INCR
        $DC 7272
        $DC 5160
        $DC 5766
        $DC 7632
        $DC 7437
        $DC 1720
        $DC 0
        POP
D13     INCR
        $DC 7272
        $DC 5270
        $DC 5756
        $DC 5564
        $DC 2057
        $DC 5655
        $DC 6430
        $DC 1720
        $DC 0
        POP
D14     INCR
        $DC 1272
        $DC 6251
        $DC 6057
```

```
        SDC 3656
        SDC 5564
        SDC 3020
        SDC 1700
        POP
D15     INCR
        SDC 7272
        SDC 5260
        SDC 6766
        SDC 6554
        SDC 3020
        SDC 1700
        POP
D16     INCR
        SDC 7272
        SDC 5270
        SDC 5025
        SDC 5550
        SDC 2516
        SDC 7050
        SDC 1720
        SDC 0
        POP
D17     INCR
        SDC 7272
        SDC 5270
        SDC 5025
        SDC 5550
        SDC 3717
        SDC 1000
        POP
D20     INCR
        SDC 1272
        SDC 6251
        SDC 6057
        SDC 3570
        SDC 5655
        SDC 6430
        SDC 1720
        SDC 0
```

```
        POP
D21     INCR
        $DC 7272
        $DC 5236
        $DC 7050
        $DC 7236
        $DC 7620
        $DC 1700
        POP
D22     INCR
        $DC 5160
        $DC 1472
        $DC 7254
        $DC 1050
        $DC 3736
        $DC 1700
        POP
D23     INCR
        $DC 2252
        $DC 5657
        $DC 6051
        $DC 7262
        $DC 3636
        $DC 1720
        $DC 0
        POP
D24     INCR
        $DC 7272
        $DC 5230
        $DC 5075
        $DC 7720
        $DC 1700
        POP
D25     INCR
        $DC 7272
        $DC 5236
        $DC 3670
        $DC 5020
        $DC 1700
        POP
```

```
D26     INCR
        SDC 7272
        SDC 5267
        SDC 6176
        SDC 7620
        SDC 1700
        POP
D27     INCR
        SDC 7272
        SDC 5277
        SDC 3250
        SDC 7676
        SDC 1720
        SDC 0
        POP
D30     INCR
        SDC 1272
        SDC 6251
        SDC 6057
        SDC 7656
        SDC 5564
        SDC 3020
        SDC 1700
        POP
D31     INCR
        SDC 7272
        SDC 5270
        SDC 5756
        SDC 5564
        SDC 3720
        SDC 1700
        POP
D32     INCR
        SDC 1272
        SDC 6251
        SDC 6057
        SDC 7656
        SDC 5564
        SDC 1022
        SDC 7720
```

```
        $DC 0
        POP
D33     INCR
        $DC 7272
        $DC 5270
        $DC 5756
        $DC 5564
        $DC 7720
        $DC 1700
        POP
D34     INCR
        $DC 1252
        $DC 5760
        $DC 5152
        $DC 5364
        $DC 5352
        $DC 5160
        $DC 5737
        $DC 3600
        POP
D35     INCR
        $DC 1150
        $DC 7272
        $DC 6420
        $DC 6036
        $DC 1637
        $DC 0
        POP
D36     INCR
        $DC 1272
        $DC 7230
        $DC 5076
        $DC 6655
        $DC 6430
        $DC 1720
        $DC 0
        POP
D37     INCR
        $DC 2272
        $DC 6230
```

```
        SDC 5076
        SDC 5665
        SDC 5327
        SDC 3010
        SDC 0
        POP
D40     INCR
        SDC 7272
        SDC 5230
        SDC 5076
        SDC 7663
        SDC 5527
        SDC 3010
        SDC 0
        POP
D41     INCR
        SDC 6271
        SDC 5152
        SDC 3454
        SDC 5657
        SDC 1767
        SDC 5617
        SDC 2000
        POP
D42     INCR
        SDC 1150
        SDC 7261
        SDC 5234
        SDC 5456
        SDC 5737
        SDC 2710
        SDC 0
        POP
D43     INCR
        SDC 6271
        SDC 5152
        SDC 7454
        SDC 3727
        SDC 5574
        SDC 3020
```

```
        $DC 1700
        POP
D44     INCR
        $DC 1251
        $DC 7151
        $DC 3454
        $DC 5717
        $DC 6727
        $DC 0
        POP
D45     INCR
        $DC 1252
        $DC 7151
        $DC 3736
        $DC 0
        POP
D46     INCR
        $DC 1151
        $DC 7252
        $DC 1555
        $DC 5010
        $DC 6037
        $DC 1600
        POP
D47     INCR
        $DC 3252
        $DC 7050
        $DC 3716
        $DC 0
        POP
D50     INCR
        $DC 1150
        $DC 5372
        $DC 5251
        $DC 3727
        $DC 2600
        POP
D51     INCR
        $DC 1150
        $DC 5172
```

```
        $DC 5253
        $DC 3727
        $DC 2600
        POP
D52     INCR
        $DC 5172
        $DC 7260
        $DC 3636
        $DC 5450
        $DC 1730
        $DC 0
        POP
D53     INCR
        $DC 5160
        $DC 7272
        $DC 6437
        $DC 3716
        $DC 0
        POP
D54     INCR
        $DC 3051
        $DC 7371
        $DC 3736
        $DC 1600
        POP
D55     INCR
        $DC 3150
        $DC 7454
        $DC 1252
        $DC 7050
        $DC 3706
        $DC 0
        POP
D56     INCR
        $DC 5271
        $DC 7337
        $DC 3717
        $DC 0
        POP
D57     INCR
```

```
        $DC 1150
        $DC 7272
        $DC 6520
        $DC 5157
        $DC 3726
        $DC 0
        POP
D60     INCR
        $DC 1151
        $DC 6361
        $DC 1655
        $DC 7037
        $DC 1600
        POP
D61     INCR
        $DC 6250
        $DC 5630
        $DC 1720
        $DC 0
        POP
D62     INCR
        $DC 6250
        $DC 6655
        $DC 1130
        $DC 3000
        POP
D63     INCR
        $DC 1262
        $DC 1262
        $DC 5056
        $DC 1666
        $DC 2710
        $DC 3000
        POP
D64     INCR
        $DC 1262
        $DC 1262
        $DC 5056
        $DC 1676
        $DC 5530
```

```
        $DC 3010
        $DC 0
        POP
D65     INCR
        $DC 1150
        $DC 1262
        $DC 5051
        $DC 5253
        $DC 6455
        $DC 3737
        $DC 1000
        POP
D66     INCR
        $DC 1150
        $DC 1272
        $DC 6237
        $DC 2726
        $DC 0
        POP
D67     INCR
        $DC 2122
        $DC 7237
        $DC 2726
        $DC 0
        POP
D70     INCR
        $DC 1132
        $DC 7210
        $DC 5066
        $DC 3717
        $DC 1600
        POP
D71     INCR
        $DC 1252
        $DC 5760
        $DC 5152
        $DC 5364
        $DC 5352
        $DC 5160
        $DC 5714
```

```
        $DC 5456
        $DC 1666
        $DC 2730
        $DC 0
        POP
D72     INCR
        $DC 5172
        $DC 7210
        $DC 5076
        $DC 7612
        $DC 5314
        $DC 5412
        $DC 5210
        $DC 5010
        $DC 5016
        $DC 5637
        $DC 0
        POP
D73     INCR
        $DC 5153
        $DC 5261
        $DC 5355
        $DC 1777
        $DC 1454
        $DC 1151
        $DC 3700
        POP
D75     VEC
        $DC 2020
        $DC 4000
        POP
D76     SVEC
        $DC 50
        POP
```

```
        STITLE                  TRACKING PATTERN GENERATOR

XP      LAW 3000
        SDC 1105
XPY     SDC 1000
XPX     SDC 5000
        SDC 1400
        SDC X1
        SDC 60
        SVEC
        SDC 24
        SDC 4047
        SDC 1400
        SDC X2
        SDC 60
        SVEC
        SDC 1
        SDC 4067
        SDC 1400
        SDC X3
        SDC 60
        SVEC
        SDC 2403
        SDC 4740
        SDC 1400
        SDC X2
        SDC 60
        SVEC
        SDC 100
        SDC 6700
        SDC 340
        SDC 1400
        SDC X4
XPS     POP
        SVEC
        SDC 404
        SDC 4030
        SDC 7000
        SDC 4010
        SDC 5040
```

```
VEC
$DC  4
$DC  4
$DC  4000
$DC  2020
$DC  6020
$DC  0
$DC  4000
$DC  20
$DC  4020
$DC  0
$DC  4
$DC  4
$DC  4000
$DC  2030
$DC  6030
$DC  0
$DC  4000
$DC  30
$DC  4030
$DC  0
$DC  4
$DC  4
$DC  4000
$DC  2040
$DC  6040
$DC  0
$DC  4000
$DC  40
$DC  4040
$DC  0
$DC  4
$DC  4
$DC  4000
$DC  2050
$DC  6050
$DC  0
$DC  4000
$DC  50
$DC  4050
```

```
$DC 0
$DC 4
$DC 4
$DC 4000
$DC 2060
$DC 6060
$DC 0
$DC 4000
$DC 60
$DC 4060
$DC 0
$DC 4
$DC 4
$DC 4000
$DC 2070
$DC 6070
$DC 0
$DC 4000
$DC 70
$DC 4070
$DC 0
$DC 4
$DC 4
$DC 4000
$DC 2100
$DC 6100
$DC 0
$DC 4000
$DC 100
$DC 4100
$DC 4000
$DC 1400
$DC X5
POP
```

```
        $TITLE          INTERRUPT DISPATCHER

1       DAC 6           SAVE AC CONTENTS
        LACQ            GET MQ CONTENTS
        DAC 3           SAVE MQ CONTENTS
        LACS            GET SC CONTENTS
        DAC 2           SAVE SC CONTENTS
        IOT 1441        SKIP ON DATAPHONE RECEIVE FLAG
        SKP             TEST NEXT FLAG
        JMP IFI         SERVICE DATAPHONE INPUT INTERRUPT
        IOT 1401        SKIP ON DATAPHONE TRANSMIT FLAG
        SKP             TEST NEXT FLAG
        JMP IFO         SERVICE DATAPHONE OUTPUT INTERRUPT
        IOT 101         SKIP ON READER FLAG
        SKP             TEST NEXT FLAG
        JMP IRD         SERVICE READER INTERRUPT
        IOT 1301        SKIP ON A/D CONVERTER FLAG
        SKP             TEST NEXT FLAG
        JMP IAD         SERVICE A/D CONVERTER INTERRUPT
        IOT 301         SKIP ON KEYBOARD FLAG
        SKP             TEST NEXT FLAG
        JMP IKB         SERVICE KEYBOARD INTERRUPT
        IOT 201         SKIP ON PUNCH FLAG
        SKP             TEST NEXT FLAG
        JMP IPC         SERVICE PUNCH INTERRUPT
        IOT 401         SKIP ON TELEPRINTER FLAG
        SKP             TEST NEXT FLAG
        JMP ITP         SERVICE TELEPRINTER INTERRUPT
        IOT 1           SKIP ON CLOCK FLAG
        SKP             TEST NEXT FLAG
        JMP ICK         SERVICE CLOCK INTERRUPT
        IOT 612         READ DISPLAY STATUS
        DAC DSS         SAVE DISPLAY STATUS WORD 1
        AND =20         GET PUSH BUTTON FLAG
        SZA             SKIP ON NO PUSH BUTTON FLAG
        JMP IPB         SERVICE PUSH BUTTON INTERRUPT
        IOT 702         SKIP ON EDGE FLAG
        JMP *+3         TEST NEXT FLAG
        IOT 724         RESUME DISPLAY
        JMP IR          RETURN FROM INTERRUPT
```

```
        IOT 642             SKIP ON LIGHT PEN FLAG
        SKP                 TEST NEXT FLAG
        JMP ILP             SERVICE LIGHT PEN INTERRUPT
        IOT 721             SKIP ON INTERNAL STOP FLAG
        SKP                 TEST NEXT FLAG
        JMP IIS             SERVICE INTERNAL STOP INTERRUPT
        IOT 722             SKIP ON MANUAL INTERRUPT FLAG
        JMP EI              INVALID INTERRUPT
        JMP EM              EMERGENCY REINITIALIZATION
IR      LAC 2               GET SC CONTENTS
        XOR =77             COMPLEMENT SHIFT COUNT
        TAD =640402         FORM NORM INSTRUCTION
        AND =640477         TRUNCATE CARRY
        DAC *+1             STORE NORM INSTRUCTION
        HLT                 RESTORE SC CONTENTS
        LAC 3               GET MQ CONTENTS
        LMQ                 RESTORE MQ CONTENTS
        LAC 6               RESTORE AC CONTENTS
        IOT 42              ENABLE INTERRUPTS
        IOT 3344            DEBREAK AND RESTORE
        JMP* 0              RETURN TO INTERRUPTED PROGRAM
```

```
        $TITLE                  SYSTEM DIAGNOSTICS

SYSTEM  LAW 4400                GET BREAK FIELD 1 PARAMETER
        IOT 705                 LOAD BREAK FIELD
        LAW =1400               GET ADDRESS OF INTERNAL STOP
        IOT 1605                INITIALIZE DISPLAY
        LAC =*+2                GET ADDRESS OF TEXT LIST
        JMP E                   INITIALIZE SYSTEM
        $DC 5
        $TEXT "SYSTEM RELOADED"

EE      IOT 42                  ENABLE INTERRUPTS
        LAC BP3                 GET PUNCH STATUS SWITCH
        SZA                     SKIP IF PUNCH IS IDLE
        JMP *-2                 WAIT FOR PUNCH TO FINISH
        LAC BT1                 GET TELEPRINTER STATUS SWITCH
        SZA                     SKIP IF TELEPRINTER IS IDLE
        JMP *-2                 WAIT FOR TELEPRINTER TO FINISH
        IOT 1412                READ 201 STATUS
        AND =2                  GET TRANSMIT STATE BIT
        SZA                     SKIP IF NOT TRANSMITTING
        JMP *-3                 WAIT FOR END OF TRANSMISSION
        IOT 2                   DISABLE INTERRUPTS
        LAC =*+2                GET ADDRESS OF TEXT LIST
        JMP E                   REINITIALIZE SYSTEM
        $DC 6
        $TEXT "TASK QUEUE EMPTY"

EI      LAC =*+2                GET ADDRESS OF TEXT LIST
        JMP E                   REINITIALIZE SYSTEM
        $DC 6
        $TEXT "INVALID INTERRUPT"

EM      LAC =*+2                GET ADDRESS OF TEXT LIST
        JMP E                   REINITIALIZE SYSTEM
        $DC 6
        $TEXT "MANUAL INTERRUPT"

EQ      LAC =*+2                GET ADDRESS OF TEXT LIST
        JMP E                   REINITIALIZE SYSTEM
```

```
        $DC 7
        $TEXT "TASK QUEUE OVERFLOW"

ES      DZM BP3                 CLEAR PUNCH STATUS SWITCH
        DZM BT1                 CLEAR TELEPRINTER STATUS SWITCH
        LAW 4400                GET BREAK FIELD 1 PARAMETER
        IOT 705                 LOAD BREAK FIELD
        LAW =1400               GET ADDRESS OF INTERNAL STOP
        IOT 1605                INITIALIZE DISPLAY
        LAC =*+2                GET ADDRESS OF TEXT LIST
        JMP E                   REINITIALIZE SYSTEM
        $DC 5
        $TEXT "PANEL RECOVERY"

ET      IOT 2                   DISABLE INTERRUPTS
        CLC                     LOAD AC WITH -1
        TAD 20                  ADD PROGRAM COUNTER DURING TRAP
        JMS C.B6                CONVERT TO 6-BIT CODE
        AND =7777               TRUNCATE HIGH ORDER DIGIT
        TAD =760000             USE BLANK AS HIGH ORDER CHARACTER
        DAC ET1                 STORE HIGH ORDER CHARACTERS
        LACQ                    GET LOW ORDER DIGITS
        DAC ET2                 STORE LOW ORDER DIGITS
        LAC =*+2                GET ADDRESS OF TEXT LIST
        JMP E                   REINITIALIZE SYSTEM
        $DC 13
        $TEXT "ILLEGAL INSTRUCTION AT LOC"
ET1     $DC 0
ET2     $DC 0
```

```
        $TITLE              SYSTEM INITIALIZER

E       DAC 25              SAVE ADDRESS OF DIAGNOSTIC
        IOT 7702            ENTER EXTEND MODE
        IOT 1412            READ 201 STATUS
        AND =1              GET RECEIVE STATE BIT
        SZA                 SKIP IF NOT RECEIVING
        JMP *-3             WAIT FOR END OF RECORD
        IOT 1444            CLEAR 201 INTERFACE
        LAC =440            GET TERM RDY BIT & FRAME SIZE 8
        IOT 1404            SET INITIAL 201 INTERFACE STATE
        LAC BP3             GET PUNCH STATUS SWITCH
        SNA                 SKIP IF PUNCH ACTIVE
        JMP *+3             PUNCH NOT ACTIVE
        IOT 201             SKIP ON PUNCH FLAG
        JMP *-1             WAIT FOR PUNCH FLAG
        LAC BT1             GET TELEPRINTER STATUS SWITCH
        SNA                 SKIP IF TELEPRINTER ACTIVE
        JMP *+3             TELEPRINTER NOT ACTIVE
        IOT 401             SKIP ON TELEPRINTER FLAG
        JMP *-1             WAIT FOR TELEPRINTER FLAG
        IOT 612             READ DISPLAY STATUS
        AND =7400           GET DISPLAY FLAG BITS
        SNA+CLA             SKIP IF DISPLAY STOPPED
        JMP *-3             WAIT FOR DISPLAY TO STOP
        JMS PS1             CLEAR PUSH BUTTONS
        IOT 4               DISABLE CLOCK
        IOT 3302            CLEAR ALL FLAGS
        DZM BP3             INDICATE PUNCH IDLE
        DZM BT1             INDICATE TELEPRINTER IDLE
        DZM PE+1            DISABLE OPERATION OF PUSH BUTTONS
        DZM DE+1            DISABLE DISPLAY INTERRUPTS
        DZM DWV             CLEAR TRANSLATION VALUE
        DZM NA+2            UNLOCK N.A
        DZM NC+2            UNLOCK N.C
        DZM STRD+2          UNLOCK S.TRD
        DZM STRR+2          UNLOCK S.TRR
        DZM SLY+2           UNLOCK S.LY
        DZM SLX+2           UNLOCK S.LX
        LAW 10              GET TELEPRINTER MASK
```

```
        DAC STATUS          ALLOCATE TELEPRINTER ONLY
        DAC BFTTY2          SET BFTTY ALLOCATION MASK
        LAC TQ              GET POINTER TO END OF TASK QUEUE
        DAC TQ+1            RESET INPUT POINTER
        DAC TQ+2            RESET OUTPUT POINTER
        DAC BRS             SET RECORD SEEK SWITCH
        DZM BRO             INDICATE NEW RECORD NEEDED
        LAC BPQ             GET POINTER TO END OF PUNCH BUFFER
        DAC BPQ+1           RESET INPUT POINTER
        DAC BPQ+2           RESET OUTPUT POINTER
        LAC BKQ             GET POINTER TO END OF KB BUFFER
        DAC BKQ+1           RESET INPUT POINTER
        DAC BKQ+2           RESET OUTPUT POINTER
        LAC BTQ             GET POINTER TO END OF TP BUFFER
        DAC BTQ+1           RESET INPUT POINTER
        DAC BTQ+2           RESET OUTPUT POINTER
        LAC =DN             GET ADDRESS OF NULL DISPLAY SERVICE
        DAC DPT             SET NULL LIGHT PEN SERVICE
        LAC =PN             GET ADDRESS OF NULL PB SERVICE
        DAC PTT             SET NULL PUSH BUTTON SERVICE
        LAW 3000            GET POP INSTRUCTION
        DAC XP              INHIBIT TRACKING PROCESS
        LAC =D              GET ADDRESS OF HIGHEST ACTIVE LEVEL
        DAC DHAL+7          REMOVE EVERYTHING FROM HAL
        LAW PDP1            GET ADDRESS OF PUSH DOWN LIST
        IOT 645             SET PUSH DOWN POINTER
        LAW 7763            GET INITIAL DISPLAY CONDITIONS
        IOT 665             SET INITIAL DISPLAY CONDITIONS
        LAW 4400            GET BREAK FIELD 1 PARAMETER
        IOT 705             LOAD BREAK FIELD
        LAW D               GET ADDRESS OF SYSTEM DISPLAY FILE
        IOT 1605            START DISPLAY
        LAC =BFENQ          GET ENQUIRY CHARACTER
        JMS BFENQS          INITIALIZE 201 TASKS
        IOT 42              ENABLE INTERRUPTS
        LAW BFENQ           GET ENQUIRY CHARACTER
        JMS B.FO            SEND ATTENTION INTERRUPT
        NOP                 DATA SET NOT CONNECTED
E1      DZM 26              CLEAR POINTER TO DIAGNOSTIC LEVEL
        DZM 27              CLEAR POINTER TO DIAGNOSTIC LEAF
```

```
JMS S.TL        CREATE TITLE LEAF
JMP E2          USE TELETYPE ONLY
DAC E3          SAVE POINTER TO TITLE LEAF
LAC =EF         GET ADDRESS OF TEXT LIST
JMS L.D         CREATE TITLE LEAF
JMP E2          USE TELETYPE ONLY
LMQ             SET UP PARAMETER
LAC E3          GET POINTER TO TITLE LEVEL
JMS S.TI        INSERT TITLE LEAF
JMP E2          USE TELETYPE ONLY
LAC =370        GET Y TITLE COORDINATE
LMQ             SET UP PARAMETER
LAC E3          GET POINTER TO TITLE LEVEL
JMS S.LY        SET Y TITLE COORDINATE
LAW -144        GET X TITLE COORDINATE
LMQ             SET UP PARAMETER
LAC E3          GET POINTER TO TITLE LEVEL
JMS S.LX        SET X TITLE COORDINATE
LAW 500         GET SCALE X2 PARAMETER
LMQ             SET UP PARAMETER
LAC E3          GET POINTER TO TITLE LEVEL
JMS S.LP        SET TITLE SCALE
LAC E3          GET POINTER TO TITLE LEVEL
LMQ             SET UP PARAMETER
LAC =DHAL       GET ADDRESS OF HIGHEST ACTIVE LEVEL
JMS S.TI        INSERT TITLE LEVEL
JMP E2          USE TELETYPE ONLY
JMS S.TL        CREATE DIAGNOSTIC LEVEL
JMP E2          USE TELETYPE ONLY
DAC 26          SET POINTER TO DIAGNOSTIC LEVEL
LAC =200        GET Y DIAGNOSTIC DISPLACEMENT
LMQ             SET UP PARAMETER
LAC 26          GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S.LY        TRANSLATE LEVEL IN Y DIRECTION
LAW -400        GET X DIAGNOSTIC DISPLACEMENT
LMQ             SET UP PARAMETER
LAC 26          GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S.LX        TRANSLATE LEVEL IN X DIRECTION
LAW 500         GET SCALE X2 PARAMETER
LMQ             SET UP PARAMETER
```

```
        LAC 26              GET ADDRESS OF DIAGNOSTIC LEVEL
        JMS S.LP            SET DIAGNOSTIC SCALE
        LAC 26              GET ADDRESS OF DIAGNOSTIC LEVEL
        LMQ                 SET UP PARAMETER
        LAC =DHAL           GET ADDRESS OF HIGHEST ACTIVE LEVEL
        JMS S.TI            INSERT DIAGNOSTIC LEVEL
        JMP E2              DISPLAY STORAGE EXCEEDED
        LAC 25              GET ADDRESS OF TEXT LIST
        SZA                 SKIP IF DISP STORAGE BEING CLEARED
        JMS L.D             CREATE DIAGNOSTIC LEAF
        JMP E2              USE TELETYPE ONLY
        DAC 27              SET POINTER TO DIAGNOSTIC LEAF
        LMQ                 SET UP PARAMETER
        LAC 26              GET ADDRESS OF DIAGNOSTIC LEVEL
        JMS S.TI            INSERT DIAGNOSTIC LEAF
        NOP                 USE TELETYPE ONLY
E2      LAC 25              GET POINTER TO TEXT LIST
        SNA                 SKIP IF COMMENT TO BE TYPED
        JMP IDLE            BEGIN IDLE-TIME TASK
        LAC =747575         GET TELEPRINTER POSITIONING CODE
        JMS B.T             POSITION TELEPRINTER
        LAC 25              GET ADDRESS OF TEXT LIST
        JMS L.T             TYPE DIAGNOSTIC
        LAC =747575         GET TELEPRINTER POSITIONING CODE
        JMS B.T             POSITION TELEPRINTER
        JMP IDLE            BEGIN IDLE-TIME TASK

EF      $DC 11
        $TEXT "SEL EXECUTIVE SYSTEM (01)"
```

```
        $TITLE          DISPLAY STRUCTURE STORAGE MANAGER

STORE   $EQU 12000      LOWER LIMIT OF DISPLAY STORAGE

B       $DC 0
        CMA             FORM 1'S COMP OF NUMBER OF BLOCKS
        TAD =1          FORM 2'S COMP OF NUMBER OF BLOCKS
        DAC T1          INITIALIZE COUNTER
        DAC T2          STORE VALUE FOR RESETTING COUNTER
        LAC =STORE      GET LOWER LIMIT OF DISPLAY STORAGE
B1      DAC T3          SET POINTER TO CANDIDATE
        DAC T4          SET POINTER TO NEW BLOCK
        SAD =20000      SKIP IF STORAGE NOT EXCEEDED
        JMP* B          NOT ENOUGH FREE STORAGE
        LAC* T4         GET FIRST WORD FROM BLOCK
        SNA             SKIP IF BLOCK NOT AVAILABLE
        JMP B2          ADD BLOCK TO CANDIDATE
        LAC T2          GET INITIAL VALUE OF COUNTER
        DAC T1          REINITIALIZE COUNTER
        LAC T4          GET ADDRESS OF UNAVAILABLE BLOCK
        TAD =4          FORM ADDRESS OF NEXT BLOCK
        JMP B1          PROCEED WITH NEXT CANDIDATE
B2      ISZ T1          INCREMENT COUNTER & SKIP IF DONE
        JMP *+4         PREPARE TO ADD ANOTHER BLOCK
        LAC T3          GET ADDRESS OF ACQUIRED STORAGE
        ISZ B           INDICATE SUCCESS
        JMP* B          RETURN
        LAC T4          GET ADDRESS OF BLOCK JUST ADDED
        TAD =4          FORM ADDRESS OF NEXT BLOCK
        JMP B1+1        ADD ANOTHER BLOCK

B3      $DC 0
        LAC =1          GET SINGLE BLOCK PARAMETER
        JMS B           FIND SINGLE BLOCK
        JMP* B3         NO SINGLE BLOCK AVAILABLE
        ISZ B3          INDICATE SUCCESS
        JMP* B3         RETURN

B4      $DC 0
        LAC =2          GET DOUBLE BLOCK PARAMETER
```

```
JMS B           FIND DOUBLE BLOCK
JMP* B4         NO DOUBLE BLOCK AVAILABLE
ISZ B4          INDICATE SUCCESS
JMP* B4         RETURN
```

```
        STITLE          WORD QUEUE MANAGER

QC      IOT 2           DISABLE INTERRUPTS
        JMS QS          SET CONTROL POINTERS
        LAC* QP         GET POINTER TO END OF QUEUE
        DAC* QIP        SET INPUT POINTER
        DAC* QOP        SET OUTPUT POINTER
        IOT 42          ENABLE INTERRUPTS
        JMP* Q.C        RETURN

QA      IOT 2           DISABLE INTERRUPTS
        JMS QA1         ADD WORD TO QUEUE
        SKP             OVERFLOW
        ISZ Q.A         INDICATE SUCCESS
        IOT 42          ENABLE INTERRUPTS
        JMP* Q.A        RETURN

QI      IOT 2           DISABLE INTERRUPTS
        JMS QS          SET CONTROL POINTERS
        LAC* QOP        GET OUTPUT POINTER
        DAC 23          SAVE OUTPUT POINTER
        TAD =-3         SUBTRACT 3
        SAD QP          SKIP IF NO WRAP-AROUND
        LAC* QP         GET POINTER TO END OF QUEUE
        SAD* QP         SKIP IF NO WRAP-AROUND
        SKP             CHECK FOR OVERFLOW
        TAD =2          FORM NEW OUTPUT POINTER
        SAD* QIP        SKIP IF NO OVERFLOW
        JMP *+5         OVERFLOW
        DAC* QOP        SET NEW OUTPUT POINTER
        LACQ            GET VALUE TO BE STORED
        DAC* 23         STORE VALUE IN QUEUE
        ISZ Q.I         INDICATE SUCCESS
        IOT 42          ENABLE INTERRUPTS
        JMP* Q.I        RETURN

QF      IOT 2           DISABLE INTERRUPTS
        JMS QF1         FETCH WORD FROM QUEUE
        SKP             QUEUE EMPTY
        ISZ Q.F         INDICATE SUCCESS
```

```
        IOT 42              ENABLE INTERRUPTS
        JMP* Q.F            RETURN

QA1     $DC 0
        JMS QS              SET CONTROL POINTERS
        LAC* QIP            GET INPUT POINTER
        JMS QINC            INCREMENT
        SAD* QOP            SKIP IF NO OVERFLOW
        JMP* QA1            OVERFLOW
        DAC* QIP            SET NEW INPUT POINTER
        DAC 23              SAVE COPY OF POINTER
        LACQ                GET WORD TO BE STORED
        DAC* 23             STORE WORD IN QUEUE
        ISZ QA1             INDICATE SUCCESS
        JMP* QA1            RETURN

QF1     $DC 0
        JMS QS              SET CONTROL POINTERS
        LAC* QOP            GET OUTPUT POINTER
        SAD* QIP            SKIP IF QUEUE NOT EMPTY
        JMP* QF1            QUEUE EMPTY
        JMS QINC            INCREMENT
        DAC* QOP            SET NEW OUTPUT POINTER
        DAC 23              SAVE COPY OF POINTER
        LAC* 23             GET WORD FROM QUEUE
        ISZ QF1             INDICATE SUCCESS
        JMP* QF1            RETURN

QS      $DC 0
        DAC QP              SET POINTER TO QUEUE
        TAD =1              COMPUTE ADDRESS OF NEXT LOCATION
        DAC QIP             SET POINTER TO INPUT POINTER
        TAD =1              COMPUTE ADDRESS OF NEXT LOCATION
        DAC QOP             SET POINTER TO OUTPUT POINTER
        JMP* QS             RETURN

QINC    $DC 0
        SAD* QP             SKIP IF NOT END OF QUEUE
        LAC QOP             WRAP AROUND TO BEGINNING OF QUEUE
        TAD =1              INCREMENT
```

```
JMP* QINC                RETURN
```

```
        STITLE          TASK SCHEDULER

TS      IOT 2           DISABLE INTERRUPTS
        AND =77777      TRUNCATE HIGH ORDER BITS
        JMS TII         PUT TASK ADDRESS ON QUEUE
        IOT 42          ENABLE INTERRUPTS
        JMP* T.S        RETURN

TP      LAW 17776       LOAD AC WITH -2
        TAD T.P         FORM ADDRESS OF NEW TASK
        JMS T.S         SCHEDULE NEW TASK
TF      IOT 2           DISABLE INTERRUPTS
        JMS TIO         READ WORD FROM TASK QUEUE
        DAC 23          SAVE TASK ADDRESS
        RAL             SHIFT TYPE BITS INTO LINK & SIGN
        SZL             SKIP IF NOT REENTRY DELAY
        JMP TF1         RESTORE MQ & AC AND EXECUTE
        SPA             SKIP IF NOT ALLOCATION DELAY
        JMP TF2         CHECK ELIGIBILITY
        IOT 42          ENABLE INTERRUPTS
        JMP* 23         EXECUTE TASK
TF1     JMS TIO         READ WORD FROM TASK QUEUE
        LMQ             RESTORE MQ
        JMS TIO         READ WORD FROM TASK QUEUE
        JMP TF1-2       EXECUTE TASK
TF2     JMS TIO         READ WORD FROM TASK QUEUE
        AND STATUS      FORM ELIGIBILITY CHECK
        SNA             SKIP IF TASK NOT ELIGIBLE
        JMP TF3         MODIFY STATUS & EXECUTE
        LAC 23          GET ADDRESS OF TASK
        JMS TII         PUT BACK ON TASK QUEUE
        LAC* TQ+2       GET ALLOCATION MASK
        JMS TII         PUT BACK ON TASK QUEUE
        IOT 42          ENABLE INTERRUPTS
        JMP TF          GET ANOTHER TASK
TF3     LAC* TQ+2       GET ALLOCATION MASK
        XOR STATUS      OR WITH STATUS WORD
        DAC STATUS      STORE NEW STATUS WORD
        JMP TF1-2       EXECUTE TASK
```

```
TA      AND =17777          TRUNCATE HIGH ORDER BITS
        IOT 2               DISABLE INTERRUPTS
        DAC 23              SAVE ALLOCATION MASK
        LAC T.A             GET ADDRESS OF RETURN
        AND =77777          TRUNCATE HIGH ORDER BITS
        XOR =200000         INDICATE ALLOCATION DELAY
        JMS TII             PUT TASK ADDRESS ON QUEUE
        LAC 23              GET ALLOCATION MASK
        JMS TII             PUT ALLOCATION MASK ON QUEUE
        JMP TF+1            GET ANOTHER TASK

TR      CMA                 COMPLEMENT RELEASE MASK
        AND STATUS          MODIFY ALLOCATION STATUS
        DAC STATUS          STORE NEW ALLOCATION STATUS
        JMP* T.R            RETURN

TL      DAC T1              SAVE AC CONTENTS
        LAW 17776           LOAD AC WITH -2
        TAD T.L             FORM ADDRESS OF SUBROUTINE ENTRY
        DAC T2              SAVE ADDRESS OF SUBROUTINE ENTRY
        LAC* T.L            GET SAVED RETURN POINTER
        SZA                 SKIP IF SUBROUTINE ENTERABLE
        JMP TL1             RESCHEDULE SUBROUTINE CALL
        LAC* T2             GET RETURN POINTER
        DAC* T.L            SAVE AND LOCK SUBROUTINE
        LAC T1              RESTORE AC CONTENTS
        ISZ T.L             ADVANCE PAST SAVED RETURN POINTER
        JMP* T.L            RETURN
TL1     CLC                 LOAD AC WITH -1
        TAD* T2             FORM ADDRESS OF SUBROUTINE CALL
        AND =77777          TRUNCATE HIGH ORDER BITS
        XOR =400000         INDICATE REENTRY DELAY
        IOT 2               DISABLE INTERRUPTS
        JMS TII             PUT TASK ADDRESS ON QUEUE
        LACQ                GET CONTENTS OF MQ
        JMS TII             PUT ON TASK QUEUE
        LAC T1              RESTORE AC CONTENTS
        JMS TII             PUT ON TASK QUEUE
        JMP TF+1            GET A NEW TASK
```

```
TU      DAC T1              SAVE AC CONTENTS
        LAC* T.U            GET ADDRESS OF SUBROUTINE
        TAD =2              FORM ADDRESS OF SAVED RETURN
        DAC T2              SAVE ADDRESS OF SAVED RETURN
        LAC* T2             GET SAVED RETURN
        DAC T3              SAVE TEMPORARILY
        DZM* T2             UNLOCK SUBROUTINE
        LAC T1              RESTORE AC CONTENTS
        JMP* T3             RETURN FROM SUBROUTINE

TV      $DC 0
        DAC T1              SAVE AC CONTENTS
        LAC* TV             GET POINTER TO SUBROUTINE
        DAC T2              SAVE POINTER TO SUBROUTINE
        ISZ TV              FORM POINTER TO SAVED RETURN
        LAC TV              GET POINTER TO SAVED RETURN
        DAC T.L             SIMULATE CALL TO T.L
        JMP TL+4            FAKE AN ENTRY TO T.L

TIO     $DC 0
        LAC TQ+2            GET OUTPUT POINTER
        SAD TQ+1            SKIP IF TASK QUEUE NOT EMPTY
        JMP EE              TASK QUEUE EMPTY
        JMS TI              INCREMENT
        DAC TQ+2            STORE NEW OUTPUT POINTER
        LAC* TQ+2           GET WORD FROM TASK QUEUE
        JMP* TIO            RETURN

TII     $DC 0
        DAC 24              SAVE VALUE TO BE STORED
        LAC TQ+1            GET INPUT POINTER
        JMS TI              INCREMENT
        DAC TQ+1            STORE NEW INPUT POINTER
        SAD TQ+2            SKIP IF NO TASK QUEUE OVERFLOW
        JMP EQ              TASK QUEUE OVERFLOW
        LAC 24              GET VALUE TO BE STORED
        DAC* TQ+1           PUT IN TASK QUEUE
        JMP* TII            RETURN

TI      $DC 0
```

```
        SAD TQ                  SKIP IF NO WRAP-AROUND
        LAC =TQ+2               GET ADDRESS BEFORE FIRST DATA WORD
        TAD =1                  INCREMENT POINTER
        JMP* TI                 RETURN

TQ      $DC *+200
        $DS 200
```

```
        STITLE          FORMAT CONVERTER

CB6     CLL             USE ZEROS TO FILL HOLES
        LRS 14          SHIFT DIGITS 2, 3, 4, & 5 INTO MQ
        ALS 3           CONVERT DIGIT 2
        LRS 6           SHIFT DIGIT 2 INTO MQ
        ALS 3           CONVERT DIGIT 1
        LLS 11          SHIFT DIGITS 0, 1, & 2 INTO AC
        DAC T1          STORE HIGH ORDER DIGITS
        LLS 6           SHIFT DIGITS 3 & 4 INTO AC
        ALS 3           CONVERT DIGIT 5
        LRS 6           SHIFT DIGIT 4 INTO MQ
        ALS 3           CONVERT DIGIT 4
        AND =77         CONVERT DIGIT 3
        LRS 11          SHIFT LOW ORDER DIGITS INTO MQ
        LAC T1          GET HIGH ORDER DIGITS
        JMP* C.B6       RETURN

C6A     AND =77         TRUNCATE HIGH ORDER BITS
        TAD =C6A1       ADD ADDRESS OF TABLE
        DAC T1          SAVE TEMPORARILY
        LAC* T1         GET CONVERTED VALUE
        JMP* C.6A       RETURN

C6A1    $DC 260
        $DC 261
        $DC 262
        $DC 263
        $DC 264
        $DC 265
        $DC 266
        $DC 267
        $DC 270
        $DC 271
        $DC 301
        $DC 302
        $DC 303
        $DC 304
        $DC 305
        $DC 306
```

```
$DC 307
$DC 310
$DC 311
$DC 312
$DC 313
$DC 314
$DC 315
$DC 316
$DC 317
$DC 320
$DC 321
$DC 322
$DC 323
$DC 324
$DC 325
$DC 326
$DC 327
$DC 330
$DC 331
$DC 332
$DC 252
$DC 257
$DC 253
$DC 255
$DC 250
$DC 251
$DC 333
$DC 335
$DC 274
$DC 275
$DC 276
$DC 336
$DC 337
$DC 256
$DC 254
$DC 272
$DC 273
$DC 277
$DC 241
$DC 247
```

```
        $DC 242
        $DC 244
        $DC 243
        $DC 246
        $DC 215
        $DC 212
        $DC 240
        $DC 377

CA6     AND =177            TRUNCATE HIGH ORDER BITS
        TAD =CA61           ADD ADDRESS OF TABLE
        DAC T1              SAVE TEMPORARILY
        LAC* T1             GET CONVERTED VALUE
        JMP* C.A6           RETURN

CA61    $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 75
        $DC 77
        $DC 77
        $DC 74
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
```

```
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 76
$DC 66
$DC 70
$DC 72
$DC 71
$DC 77
$DC 73
$DC 67
$DC 50
$DC 51
$DC 44
$DC 46
$DC 62
$DC 47
$DC 61
$DC 45
$DC 00
$DC 01
$DC 02
$DC 03
$DC 04
$DC 05
$DC 06
$DC 07
$DC 10
$DC 11
$DC 63
$DC 64
$DC 54
$DC 55
$DC 56
$DC 65
$DC 77
```

```
$DC 12
$DC 13
$DC 14
$DC 15
$DC 16
$DC 17
$DC 20
$DC 21
$DC 22
$DC 23
$DC 24
$DC 25
$DC 26
$DC 27
$DC 30
$DC 31
$DC 32
$DC 33
$DC 34
$DC 35
$DC 36
$DC 37
$DC 40
$DC 41
$DC 42
$DC 43
$DC 52
$DC 77
$DC 53
$DC 57
$DC 60
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
```

```
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77
        $DC 77

CCB     DAC T1              SAVE VALUE TO BE CONVERTED
        AND =1777           GET MAGNITUDE
        DAC T2              SAVE MAGNITUDE
        LAC T1              GET VALUE TO BE CONVERTED
        AND =2000           GET SIGN BIT
        SNA                 SKIP IF NEGATIVE
        JMP CCB1            DO NOT MODIFY MAGNITUDE
        LAC T2              GET MAGNITUDE
        CMA                 FORM 1'S COMPLEMENT
        TAD =1              FORM 2'S COMPLEMENT
        JMP* C.CB           RETURN
CCB1    LAC T2              GET CONVERTED VALUE
        JMP* C.CB           RETURN

CBC     SMA                 SKIP IF NEGATIVE
        JMP CBC1            DO NOT FORM NEGATIVE
```

```
        CMA             FORM 1'S COMPLEMENT
        AND =1777       GET MAGNITUDE
        TAD =2001       SET SIGN BIT & FORM 2'S COMPLEMENT
        AND =3777       CLEAR ESCAPE/INTENSITY BIT
        JMP* C.BC       RETURN
CBC1    AND =1777       CONVERT TO MODULO 2↑10
        JMP* C.BC       RETURN
```

```
        $TITLE                  201 DATAPHONE BUFFER MANAGER

BFDLE   $EQU 220                DATA LINK ESCAPE
BFSYN   $EQU 26                 SYNCHRONOUS IDLE
BFACK   $EQU 6                  POSITIVE ACKNOWLEDGEMENT
BFNAK   $EQU 225                NEGATIVE ACKNOWLEDGEMENT
BFEOT   $EQU 204                END OF TRANSMISSION
BFENQ   $EQU 5                  ENQUIRY
BFETB   $EQU 27                 END OF TEXT BLOCK
BFETX   $EQU 3                  END OF TEXT

* STATE BITS (LOW ORDER 5 BITS OF BFS):
*   01   ACK OUTSTANDING
*   02   LAST INPUT RECORD COMPLETELY RECEIVED
*   04   ACK OUTPUT PENDING
*   10   NAK OUTPUT PENDING
*   20   DATA OUTPUT PENDING

BFI     IOT 1412                READ 201 STATUS
        AND =1000               GET SET READY BIT
        SNA                     SKIP IF DATA SET CONNECTED
        JMP* B.FI               DATA SET NOT CONNECTED
        LAC BFS                 GET 201 TASK STATE
        AND =2                  GET INPUT RECORD AVAILABLE BIT
        SNA                     SKIP IF INPUT RECORD AVAILABLE
        JMP BFI2                WAIT FOR INPUT RECORD
        LAC BFIB                GET FIRST RECEIVED CHARACTER
        SZA                     SKIP IF USER RECORD
        JMP BFI2                WAIT FOR RECORD TO BE TYPED
        LAC* BFIO               GET CHARACTER FROM INPUT BUFFER
        DAC BFI3                SAVE INPUT CHARACTER
        ISZ BFIO                INCREMENT INPUT POINTER
        SMA                     SKIP IF END OF RECORD
        JMP BFI1                RETURN
        LAC BFS                 GET 201 TASK STATE
        XOR =6                  FORM ACK PENDING STATE
        DAC BFS                 SET NEW STATE
        LAC =BFXMT              GET ADDRESS OF TRANSMISSION TASK
        JMS T.S                 SCHEDULE TRANSMISSION TASK
        LAC BFI3                GET END OF RECORD CHARACTER
```

```
BFI1    ISZ B.FI            INDICATE SUCCESS
        JMP* B.FI           RETURN
        JMP BFI             GET CHARACTER FROM INPUT BUFFER
BFI2    JMS T.P             SCHEDULE PREVIOUS LOC & TERMINATE

BFO     DAC BFO3            SAVE CHARACTER TO BE BUFFERED
        IOT 1412            READ 201 STATUS
        AND =1000           GET SET READY BIT
        SNA                 SKIP IF DATA SET CONNECTED
        JMP* B.FO           DATA SET NOT CONNECTED
        LAC BFS             GET 201 TASK STATE
        AND =21             GET DATA OUTPUT & ACK EXP BITS
        SZA                 SKIP IF OUTPUT BUFFER IS FREE
        JMP BFO2            PUT CHARACTER INTO BUFFER LATER
        LAC BFO3            GET CHARACTER TO BE BUFFERED
        DAC* BFOI           PUT CHARACTER IN OUTPUT BUFFER
        ISZ BFOI            INCREMENT INPUT POINTER
        SMA                 SKIP IF END-OF-RECORD CHARACTER
        JMP BFO1            RETURN
        LAC BFS             GET 201 TASK STATE
        XOR =20             SET DATA OUTPUT PENDING BIT
        DAC BFS             SET NEW 201 TASK STATE
        LAC =BFXMT          GET ADDRESS OF TRANSMISSION TASK
        JMS T.S             SCHEDULE TRANSMISSION TASK
BFO1    ISZ B.FO            INDICATE SUCCESS
        JMP* B.FO           RETURN
        JMP BFO+1           PUT CHARACTER IN BUFFER
BFO2    JMS T.P             SCHEDULE PREVIOUS LOC & TERMINATE

BFXMT   IOT 1412            READ 201 STATUS
        AND =60100          GET CAR DET, XMT REQ, CLR SEND BITS
        SZA                 SKIP IF ABLE TO TRANSMIT
        JMP BFXMT4          RESCHEDULE BFXMT
        LAC BFS             GET 201 TASK STATE
        RAR                 SHIFT ACK EXPECTED BIT INTO LINK
        SZL+RAR             SKIP IF ACK NOT EXPECTED
        JMP BFXMT4          RESCHEDULE BFXMT
        SZL+RAR             SKIP IF INPUT BUFFER EMPTY
        JMP BFXMT4          RESCHEDULE BFXMT
        SNL+RAR             SKIP IF ACK OUTPUT PENDING
```

```
        JMP BFXMT1          CHECK FOR NAK OUTPUT PENDING
        LAC BFS             GET 201 TASK STATE
        AND =33             CLEAR ACK OUTPUT PENDING BIT
        DAC BFS             SET NEW 201 TASK STATE
        LAC =BFIB           GET ADDRESS OF INPUT BUFFER
        DAC BFII            RESET INPUT POINTER
        DAC BFIO            RESET OUTPUT POINTER
        LAC =BFACKR         GET POINTER TO ACK RECORD
        JMP BFXMT3          TRANSMIT ACK
BFXMT1  SNL+RAR             SKIP IF NAK OUTPUT PENDING
        JMP BFXMT2          CHECK FOR DATA OUTPUT PENDING
        LAC BFS             GET 201 TASK STATE
        AND =27             CLEAR NAK OUTPUT PENDING BIT
        DAC BFS             SET NEW 201 TASK STATE
        LAC =BFNAKR         GET POINTER TO NAK RECORD
        JMP BFXMT3          TRANSMIT NAK
BFXMT2  SNL                 SKIP IF DATA OUTPUT PENDING
        JMS T.F             NO OUTPUT PENDING
        LAC BFS             GET 201 TASK STATE
        XOR =21             CLEAR DATA BIT, SET ACK EXP BIT
        DAC BFS             SET NEW 201 TASK STATE
        LAC =BFOB           GET POINTER TO OUTPUT BUFFER
BFXMT3  DAC BFOO            SET OUTPUT POINTER
        LAW -10             LOAD AC WITH -8
        DAC BFC             SET SYN COUNT
        LAW BFSYN           GET SYN CHARACTER
        DAC 5               SET TRANSMIT IMAGE
        LAC =200005         GET LAC 5 INSTRUCTION
        DAC IFO             INITIALIZE XMT INTERRUPT SERVICE
        LAC =20000          GET XMT REQ BIT MASK
        IOT 1404            SET XMT REQ BIT
        JMS T.F             TERMINATE
        JMP BFXMT           START TRANSMISSION, IF APPLICABLE
BFXMT4  JMS T.P             SCHEDULE PREVIOUS LOC & TERMINATE

BFACKR  LAW BFACK           ACK RECORD
BFNAKR  LAW BFNAK           NAK RECORD

BFTTY   LAC BFTTY2          GET CONDITIONAL TELEPRINTER MASK
        JMS T.A             ALLOCATE TELEPRINTER, IF NECESSARY
```

```
        DZM BFTTY2          PREPARE FOR POSSIBLE ENQUIRY
        LAC* BFIO           GET CHARACTER FROM BUFFER
        SPA                 SKIP IF NOT END-OF-RECORD CHARACTER
        JMP BFTTY1          TERMINATE LINE
        JMS C.A6            CONVERT TO 6-BIT CODE
        XOR =777700         PRECEDE WITH NULL CHARACTERS
        JMS B.T             TYPE CHARACTER
        ISZ BFIO            INCREMENT INPUT POINTER
        JMP BFTTY+3         TYPE NEXT CHARACTER
BFTTY1  LAW 17475           GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T             TYPE CARRIAGE RETURN, LINE FEED
        LAC BFS             GET 201 TASK STATE
        XOR =6              FORM ACK PENDING STATE
        DAC BFS             SET NEW 201 TASK STATE
        LAW 10              GET ALLOCATION MASK
        DAC BFTTY2          SET BFTTY ALLOACTION MASK
        JMS T.R             RELEASE TELEPRINTER
        JMP BFXMT           ACKNOWLEDGE RECORD

IFI     LAC 4               GET RECEIVED CHARACTER
        LRS 4               SHIFT INTO POSITION
        AND =377            TRUNCATE HIGH ORDER BITS
        HLT                 STATE VARIABLE
        SAD =BFSYN          SKIP IF NOT SYN
        SKP                 FIND NEXT SYN & CHANGE STATE
        JMP IFI6            IGNORE CHARACTER
        IOT 1412            READ 201 STATUS
        AND =2000           GET TEXT BIT
        IOT 1404            CLEAR TEXT BIT
        LAC =600000+IFI1    GET JMP IFI1 INSTRUCTION
        DAC IFI+3           MODIFY INTERRUPT SERVICE
        IOT 1442            CLEAR 201 FLAGS
        JMP IR              RETURN FROM INTERRUPT
IFI1    SAD =BFSYN          SKIP IF NOT SYN
        JMP IFI1-2          IGNORE SYN
        SAD =BFDLE          SKIP IF NOT DLE (EVEN PARITY)
        JMP *+3             BUFFER RECEIVED RECORD
        LAC =740000         GET NOP INSTRUCTION
        JMP IFI1-3          MODIFY INTERRUPT SERVICE
        LAC =600000+IFI2    GET JMP IFI2 INSTRUCTION
```

```
        JMP  IFI1-3           MODIFY INTERRUPT SERVICE
IFI2    SAD  =BFDLE           SKIP IF NOT DLE (EVEN PARITY)
        JMP  IFI3-2           CHANGE STATE FOR NEXT CHARACTER
        SAD  =BFDLE-200       SKIP IF NOT DLE (ODD PARITY)
        JMP  IFI3-2           CHANGE STATE FOR NEXT CHARACTER
        JMS  BFIS             PUT CHARACTER IN BUFFER
        JMP  IFI1-2           CLEAR FLAGS AND RETURN
        LAC  =600000+IFI3     GET JMP IFI3 INSTRUCTION
        JMP  IFI1-3           MODIFY INTERRUPT SERVICE
IFI3    SAD  =BFDLE           SKIP IF NOT DLE (EVEN PARITY)
        JMP  IFI31-3          PUT DLE IN BUFFER
        SAD  =BFDLE-200       SKIP IF NOT DLE (ODD PARITY)
        JMP  IFI31-3          PUT DLE IN BUFFER
        SAD  =BFSYN           SKIP IF NOT SYN
        JMP  IFI31-2          IGNORE SYN
        DAC  BFEOR            SAVE END-OF-RECORD CHARACTER
        SAD  =BFACK           SKIP IF NOT ACK
        JMP  IFI31            CLEAR OUTPUT BUFFER
        SAD  =BFNAK           SKIP IF NOT NAK
        JMP  IFI32            RETRANSMIT LAST DATA RECORD
        XOR  =760000          INDICATE END-OF-RECORD CHARACTER
        JMS  BFIS             PUT CHARACTER IN BUFFER
        LAC  =600000+IFI4     GET JMP IFI4 INSTRUCTION
        JMP  IFI1-3           MODIFY INTERRUPT SERVICE
        JMS  BFIS             PUT DLE CHARACTER IN BUFFER
        LAC  =600000+IFI2     GET JMP IFI2 INSTRUCTION
        JMP  IFI1-3           MODIFY INTERRUPT SERVICE
IFI31   LAC  =BFOB            GET ADDRESS OF OUTPUT BUFFER
        DAC  BFOI             RESET INPUT POINTER
        LAC  BFS              GET 201 TASK STATE
        AND  =36              INDICATE ACK NOT EXPECTED
        DAC  BFS              STORE NEW TASK STATE
        LAC  =740000          GET NOP INSTRUCTION
        JMP  IFI1-3           MODIFY INTERRUPT SERVICE
IFI32   LAC  BFS              GET 201 TASK STATE
        XOR  =21              FORM STATE FOR RETRANSMISSION
        DAC  BFS              STORE NEW TASK STATE
        LAC  =BFXMT           GET ADDRESS OF TRANSMISSION TASK
        JMS  TII              SCHEDULE TRANSMISSION
        JMP  IFI32-2          MODIFY INTERRUPT SERVICE
```

```
IFI4    CLL                 PREPARE TO SHIFT ZEROS INTO AC
        ALS 10              SHIFT HIGH ORDER CHECK INTO POSITION
        DAC BFCKR           SAVE HIGH ORDER BLOCK CHECK
        LAC =600000+IFI5    GET JMP IFI5 INSTRUCTION
        JMP IFI1-3          MODIFY INTERRUPT SERVICE
IFI5    XOR BFCKR           FORM COMPLETE BLOCK CHECK
        SAD BFCK            SKIP IF BAD RECORD
        JMP IFI51           INDICATE INPUT BUFFER FULL
        LAC =BFIB           GET ADDRESS OF INPUT BUFFER
        DAC BFII            RESET INPUT POINTER
        DAC BFIO            RESET OUTPUT POINTER
        LAC BFS             GET 201 TASK STATE
        XOR =10             INDICATE NAK PENDING
        DAC BFS             SET NEW 201 TASK STATE
        JMP IFI4-3          SCHEDULE TRANSMISSION TASK
IFI51   LAC BFS             GET 201 TASK STATE
        XOR =2              INDICATE INPUT BUFFER FULL
        DAC BFS             SET NEW 201 TASK STATE
        LAC BFEOR           GET END-OF-RECORD CHARACTER
        JMS BFENQS          PROCESS ENQUIRY, IF PRESENT
        LAC BFIB            GET FIRST RECEIVED CHARACTER
        SNA                 SKIP IF UNSOLICITED RECORD
        JMP IFI32-2         MODIFY INTERRUPT SERVICE
        LAC =BFTTY          GET ADDRESS OF BYPASS TASK
        JMP IFI4-2          SCHEDULE BYPASS TASK
IFI6    IOT 1412            READ 201 STATUS
        AND =2000           GET TEXT BIT
        IOT 1404            CLEAR TEXT BIT
        JMP IFI1-2          CLEAR FLAGS AND RETURN

IFO     HLT                 STATE VARIABLE
        SAD =760000+BFSYN   SKIP IF NOT SYN
        JMP IFO1            SYN SENT LAST TIME
        SAD =760000+BFDLE   SKIP IF NOT DLE
        JMP IFO3            DLE SENT LAST TIME
        SMA                 SKIP IF END-OF-RECORD CHARACTER
        JMP IFO4            TEXT CHARACTER SENT LAST TIME
        JMP IFO5            ENTER BLOCK CHECK PROCEDURE
IFO1    ISZ BFC             SKIP IF LAST SYN SENT
        JMP IFO2+2          CLEAR FLAGS AND RETURN
```

```
        LAC =BFDLE          GET INITIAL DLE
        SKP                 SET TRANSMIT IMAGE
IF02    LAW BFDLE           GET DLE CHARACTER
        DAC 5               SET TRANSMIT IMAGE
        IOT 1442            CLEAR 201 FLAGS
        JMP IR              RETURN FROM INTERRUPT
IF03    LAC* BFOO           GET CHARACTER FROM BUFFER
        JMS BFCKS           UPDATE BLOCK CHECK
        LAC BFOO            GET OUTPUT POINTER
        SAD =BFOB           SKIP IF NOT FIRST CHARACTER
        DZM BFCK            CLEAR BLOCK CHECK
        LAC* BFOO           GET CHARACTER FROM OUTPUT BUFFER
        ISZ BFOO            INCREMENT OUTPUT POINTER
        JMP IF02+1          TRANSMIT CHARACTER
IF04    LAC* BFOO           GET CHARACTER FROM BUFFER
        SAD =BFDLE          SKIP IF NOT DLE (EVEN PARITY)
        JMP IF02            PRECEDE WITH DLE
        SAD =BFDLE-200      SKIP IF NOT DLE (ODD PARITY)
        JMP IF02            PRECEDE WITH DLE
        SMA                 SKIP IF END OF RECORD
        JMP IF03+1          SEND CHARACTER FROM BUFFER
        AND =377            TRUNCATE HIGH ORDER BITS
        JMS BFENQS          PROCESS ENQUIRY, IF PRESENT
        JMP IF02            PRECEDE WITH DLE
IF05    LAC =600000+IF06    GET JMP IF06 INSTRUCTION
        DAC IFO             MODIFY INTERRUPT SERVICE
        LAC BFCK            GET BLOCK CHECK
        LRS 10              SHIFT HIGH ORDER PART INTO POSITION
        JMP IF02+1          TRANSMIT HIGH ORDER BLOCK CHECK
IF06    LAC =600000+IF07    GET JMP INSTRUCTION
        DAC IFO             MODIFY INTERRUPT SERVICE
        LAC BFCK            GET BLOCK CHECK
        JMP IF02+1          TRANSMIT LOW ORDER PART
IF07    LAC =600000+IF08    GET JMP IF08 INSTRUCTION
        DAC IFO             MODIFY INTERRUPT SERVICE
        CLC                 GET PAD CHARACTER
        JMP IF02+1          TRANSMIT PAD
IF08    IOT 1412            READ 201 STATUS
        AND =20000          GET XMT REQ BIT
        IOT 1404            CLEAR XMT REQ BIT
```

```
        JMP IF02+2              CLEAR 201 FLAGS AND RETURN

BFENQS  SDC 0
        SAD =BFENQ              SKIP IF NOT ENQUIRY
        JMP *+4                 PROCESS ENQUIRY
        SAD =BFEOT              SKIP IF NOT END-OF-TRANSMISSION
        SKP                     REGARD AS ENQUIRY
        JMP* BFENQS             RETURN
        LAC =BFOB               GET ADDRESS OF OUTPUT BUFFER
        DAC BFOI                RESET INPUT POINTER
        LAC =BFIB               GET ADDRESS OF INPUT BUFFER
        DAC BFII                RESET INPUT POINTER
        DAC BFIO                RESET OUTPUT POINTER
        DZM BFIB                DO NOT SCHEDULE BYPASS TASK
        DZM BFS                 STOP 201 TASK ACTIVITY
        LAC =740000             GET NOP INSTRUCTION
        DAC IFI+3               MODIFY INTERRUPT SERVICE
        JMP* BFENQS             RETURN

BFIS    SDC 0
        DAC* BFII               PUT CHARACTER IN INPUT BUFFER
        JMS BFCKS               UPDATE BLOCK CHECK
        LAC BFII                GET INPUT POINTER
        SAD =BFIB               SKIP IF INPUT BUFFER NON-EMPTY
        DZM BFCK                CLEAR BLOCK CHECK
        ISZ BFII                INCREMENT INPUT POINTER
        JMP* BFIS               RETURN

BFCKS   SDC 0
        DAC 23                  SAVE CHARACTER
        LAW -10                 LOAD AC WITH -8
        DAC 24                  SET COUNTER
        LAC BFCK                GET FORMER BLOCK CHECK
BFCKS1  RCR                     ROTATE LOW ORDER BIT INTO LINK
        DAC BFCK                STORE NEW LOW ORDER 15 BITS
        CLQ                     PREPARE TO GET LOW ORDER CHAR BIT
        LAC 23                  GET CHARACTER REMAINS
        LRS 1                   SHIFT LOW ORDER BIT INTO MQ
        DAC 23                  STORE CHARACTER REMAINS
        LACQ                    GET LOW ORDER CHARACTER BIT
```

```
        SZA                 SKIP IF NOT SET
        CML                 OR CHECK BIT WITH CHARACTER BIT
        LAC BFCK            GET LOW ORDER 15 BITS OF CHECK
        SZL                 SKIP IF LOW ORDER BIT WAS 0
        XOR =120001         INVERT FEEDBACK BITS
        ISZ 24              INCREMENT COUNT & SKIP IF DONE
        JMP BFCKS1          PROCESS NEXT CHARACTER BIT
        DAC BFCK            STORE NEW BLOCK CHECK
        JMP* BFCKS          RETURN

BFIB    $DS 200

BFOB    $DS 200
```

```
        $TITLE          READER BUFFER MANAGER

BR      LAC BRO         GET OUTPUT POINTER
        SNA             SKIP IF NOT START OF NEW RECORD
        JMP BR2         CLEAR BUFFER & START READER
        SAD BRI         SKIP IF BUFFER NOT EMPTY
        JMP BR1         WAIT FOR MORE INPUT
        LAC* BRO        GET IMAGE FROM BUFFER
        ISZ BRO         INCREMENT OUTPUT POINTER
        SNA             SKIP IF NOT END OF RECORD
        DZM BRO         INDICATE NEW RECORD NEEDED
        ISZ B.R         INDICATE SUCCESS
        JMP* B.R        RETURN
BR1     SAD =BRQ+200    SKIP IF NOT END OF BUFFER
        JMP BR2         CLEAR BUFFER & START READER
        IOT 314         READ STATUS
        AND =1000       GET READER OUT-OF-TAPE FLAG
        SNA             SKIP IF READER OUT OF TAPE
        JMP BR2-1       SCHEDULE NEW ATTEMPT
        DZM BRO         INDICATE NEW RECORD NEEDED
        JMP* B.R        RETURN
        JMP BR          TRY AGAIN TO GET IMAGE
        JMS T.P         SCHEDULE NEW ATTEMPT
BR2     LAC =BRQ        GET ADDRESS OF READER BUFFER
        DAC BRI         SET INPUT POINTER
        DAC BRO         SET OUTPUT POINTER
        IOT 104         SELECT READER
        JMP BR2-1       SCHEDULE NEW ATTEMPT

IRD     IOT 314         READ STATUS
        AND =1000       GET READER OUT-OF-TAPE FLAG
        SZA             SKIP IF TAPE IS IN READER
        JMP IRD1        READER OUT OF TAPE
        IOT 112         READ READER BUFFER
        SZA             SKIP IF BLANK TAPE
        JMP IRD2        PUT IMAGE IN BUFFER
        LAC BRS         GET RECORD SEEK SWITCH
        SZA             SKIP IF END OF RECORD
        JMP IRD3        IGNORE BLANK TAPE
        JMS BRS         SET RECORD SEEK SWITCH
```

```
BRS     $DC 0               RECORD SEEK SWITCH
        DZM* BRI            STORE END-OF-RECORD IMAGE
        ISZ BRI             INCREMENT INPUT POINTER
        JMP IR              RETURN FROM INTERRUPT
IRD1    IOT 102             CLEAR READER FLAG
        JMP IR              RETURN FROM INTERRUPT
IRD2    DAC* BRI            STORE IN READER BUFFER
        ISZ BRI             INCREMENT INPUT POINTER
        DZM BRS             CLEAR RECORD SEEK SWITCH
        LAC BRI             GET INPUT POINTER
        SAD =BRQ+200        SKIP IF NOT END OF BUFFER
        JMP IR              RETURN FROM INTERRUPT
IRD3    IOT 104             SELECT READER
        JMP IR              RETURN FROM INTERRUPT

BRQ     $DS 200
```

```
        $TITLE          PUNCH BUFFER MANAGER

BP      DAC BP4         SAVE PUNCH IMAGE
        JMS BP2         START PUNCH, IF POSSIBLE
        LAC BP4         GET PUNCH IMAGE
        LMQ             SET UP PUNCH IMAGE AS PARAMETER
        LAC =BPQ        GET ADDRESS OF PUNCH BUFFER
        JMS Q.A         PUT PUNCH IMAGE IN BUFFER
        JMP BP1         PUNCH BUFFER FULL
        JMS BP2         START PUNCH, IF POSSIBLE
        ISZ B.P         INDICATE SUCCESS
        JMP* B.P        RETURN
BP1     IOT 314         READ STATUS
        AND =400        GET PUNCH OUT-OF-TAPE FLAG
        SZA             SKIP IF PUNCH CONTAINS TAPE
        JMP* B.P        PUNCH OUT OF TAPE
        SKP             PREPARE TO SCHEDULE NEXT LOCATION
        JMP BP+2        TRY AGAIN TO PUT IMAGE IN BUFFER
        JMS T.P         SCHEDULE PREVIOUS LOC & TERMINATE

BP2     $DC 0
        LAC BP3         GET PUNCH STATUS SWITCH
        SZA             SKIP IF PUNCH IS IDLE
        JMP* BP2        PUNCH IS ACTIVE
        LAC =BPQ        GET ADDRESS OF PUNCH BUFFER
        JMS Q.F         FETCH IMAGE FROM BUFFER
        JMP* BP2        PUNCH BUFFER EMPTY
        IOT 204         SELECT PUNCH
        JMS BP3         SET PUNCH STATUS SWITCH
BP3     $DC 0           PUNCH STATUS SWITCH
        JMP* BP2        RETURN

IPC     IOT 314         READ STATUS
        AND =400        GET PUNCH OUT-OF-TAPE FLAG
        SZA             SKIP IF PUNCH CONTAINS TAPE
        JMP IPC1        PUNCH OUT OF TAPE
        LAC =BPQ        GET ADDRESS OF PUNCH BUFFER
        JMS QF1         GET IMAGE FROM PUNCH BUFFER
        JMP IPC1        PUNCH BUFFER EMPTY
        IOT 204         SELECT PUNCH
```

```
        JMP IR              RETURN FROM INTERRUPT
IPC1    IOT 202             CLEAR PUNCH FLAG
        DZM BP3             INDICATE PUNCH IDLE
        JMP IR              RETURN FROM INTERRUPT

BP0     $DC *+100
        $DS 100
```

```
        STITLE                  KEYBOARD BUFFER MANAGER

BK      LAC =BKQ                GET ADDRESS OF KEYBOARD BUFFER
        JMS Q.F                 GET CHARACTER FROM KEYBOARD BUFFER
        JMP BK1                 WAIT FOR MORE INPUT
        DAC BKF                 SAVE ASCII FOR SYSTEM USAGE
        JMS C.A6                CONVERT TO 6-BIT CODE
        JMP* B.K                RETURN
        JMP BK                  TRY AGAIN TO RETURN CHARACTER
BK1     JMS T.P                 SCHEDULE NEW ATTEMPT

IKB     IOT 312                 READ KEYBOARD BUFFER
        LMQ                     SET UP PARAMETER
        LAC =BKQ                GET ADDRESS OF KEYBOARD BUFFER
        JMS QA1                 PUT CHARACTER IN BUFFER
        NOP                     BUFFER FULL -- IGNORE CHARACTER
        JMP IR                  RETURN FROM INTERRUPT

BKQ     SDC .+100
        SDS 100
```

```
        $TITLE          TELEPRINTER BUFFER MANAGER

BT      DAC BTS         SAVE TEMPORARILY
        LAC BTS         GET PACKED WORD TO BE BUFFERED
        LMQ             SET UP PARAMETER
        LAC =BTQ        GET ADDRESS OF TELEPRINTER BUFFER
        JMS Q.A         PUT PACKED WORD INTO TP BUFFER
        JMP BT2         TRY AGAIN LATER
        LAC BT1         GET TELEPRINTER STATUS SWITCH
        SZA             SKIP IF TELEPRINTER IDLE
        JMP* B.T        RETURN
        IOT 2           DISABLE INTERRUPTS
        LAC B.T         GET RETURN ADDRESS
        DAC 0           STORE INTERRUPT RETURN
        JMS BT1         SET TELEPRINTER STATUS SWITCH
BT1     $DC 0           TELEPRINTER STATUS SWITCH
        JMP ITP         FAKE A TELEPRINTER INTERRUPT
        JMP BT+1        TRY AGAIN TO PUT CHAR IN BUFFER
BT2     JMS T.P         SCHEDULE NEW ATTEMPT

BT3     $DC 77
BT4     $DC 77

ITP     LAC BT4         GET SECOND CHARACTER
        SAD =77         SKIP IF NOT NULL CHARACTER
        SKP             LOOK AT THIRD CHARACTER
        JMP ITP3        TYPE SECOND CHARACTER
        LAC BT3         GET THIRD CHARACTER
        SAD =77         SKIP IF NOT NULL CHARACTER
        SKP             TYPE FIRST CHARACTER
        JMP ITP4        TYPE THIRD CHARACTER
        LAC =BTQ        GET ADDRESS OF TELEPRINTER BUFFER
        JMS QF1         GET PACKED WORD FROM TP BUFFER
        JMP ITP2        CLEAR FLAG & RETURN
        DAC BT3         SET UP THIRD CHARACTER
        LRS 6           SHIFT SECOND CHARACTER INTO PLACE
        DAC BT4         SET UP SECOND CHARACTER
        LRS 6           SHIFT FIRST CHARACTER INTO PLACE
ITP1    AND =77         TRUNCATE HIGH ORDER BITS
        SAD =77         SKIP IF NOT NULL CHARACTER
```

```
        JMP  ITP          TYPE NEXT CHARACTER
        TAD  *C6A1        ADD ADDRESS OF 6-BIT TO ASCII TABLE
        DAC  23           SAVE TEMPORARILY
        LAC* 23           GET CONVERTED ASCII VALUE
        IOT  406          SEND CHARACTER TO TELEPRINTER
        JMP  IR           RETURN FROM INTERRUPT
ITP2    IOT  402          CLEAR TELEPRINTER FLAG
        DZM  BT1          INDICATE TELEPRINTER IDLE
        JMP  IR           RETURN FROM INTERRUPT
ITP3    DAC  23           SAVE TEMPORARILY
        LAC  *77          GET NULL CHARACTER
        DAC  BT4          STORE AS SECOND CHARACTER
        LAC  23           GET CHARACTER TO BE TYPED
        JMP  ITP1         TYPE SECOND CHARACTER
ITP4    DAC  23           SAVE TEMPORARILY
        LAC  *77          GET NULL CHARACTER
        DAC  BT3          STORE AS THIRD CHARACTER
        LAC  23           GET CHARACTER TO BE TYPED
        JMP  ITP1         TYPE THIRD CHARACTER

BTQ     SDC  .+100
        SDS  100
```

```
        STITLE              NONBUFFERED I/O MANAGER

NA      JMS TV              PROTECT AGAINST REENTRY
        SDC N.A
        SDC 0
        AND =77             TRUNCATE HIGH ORDER BITS
        IOT 1103            SELECT A/D CONVERTER CHANNEL
        IOT 1304            SELECT A/D CONVERTER
        DZM IAD1            CLEAR CONVERSION SWITCH
NA1     LAC IAD1            GET CONVERSION SWITCH
        SNA                 SKIP IF CONVERSION COMPLETE
        JMP NA2             WAIT FOR CONVERSION TO BE COMPLETED
        LAC NA3             GET CONVERTED VALUE
        JMS T.U             UNLOCK N.A
        SDC NA
        JMP NA1             CHECK FOR CONVERSION COMPLETE
NA2     JMS T.P             SCHEDULE CONVERSION CHECK

NC      JMS TV              PROTECT AGAINST REENTRY
        SDC N.C
        SDC 0
        DAC 7               SET CLOCK INTERVAL
        IOT 44              ENABLE CLOCK
        DZM ICK+1           CLEAR CLOCK SWITCH
        LAC ICK+1           GET CLOCK SWITCH
        SNA                 SKIP IF TIME INTERVAL HAS ELAPSED
        JMP NC1             WAIT A LITTLE LONGER
        JMS T.U             UNLOCK N.C
        SDC NC
        JMP *-5             CHECK ELAPSED TIME
NC1     JMS T.P             SCHEDULE A LATER CHECK

ND1     IOT 5101            SELECT D/A CONVERTER #1
        JMP* N.D1           RETURN

ND2     IOT 5102            SELECT D/A CONVERTER #2
        JMP* N.D2           RETURN

ND3     IOT 5104            SELECT D/A CONVERTER #3
        JMP* N.D3           RETURN
```

```
IAD     IOT 1312        READ A/D CONVERTER
        DAC NA3         STORE CONVERTED VALUE
        JMS IAD1        SET CONVERSION SWITCH
IAD1    SDC 0           CONVERSION SWITCH
        JMP IR          RETURN FROM INTERRUPT

ICK     JMS *+1         SET CLOCK SWITCH
        SDC 0           CLOCK SWITCH
        IOT 4           CLEAR CLOCK FLAG
        JMP IR          RETURN FROM INTERRUPT
```

```
        STITLE                  PUSH BUTTON PROCESSOR

PT      SNA                     SKIP IF NOT NULL TASK
        LAC =PN                 GET ADDRESS OF NULL TASK
        DAC PTT                 SAVE ADDRESS OF PUSH BUTTON SERVICE
        JMP* P.T                RETURN

PE      JMS *+1                 SET PUSH BUTTON ENABLE SWITCH
        SDC 0                   PUSH BUTTON ENABLE SWITCH
        JMP* P.E                RETURN

PD      DZM PE+1                CLEAR PUSH BUTTON ENABLE SWITCH
        JMP* P.D                RETURN

PR      IOT 631                 READ PUSH BUTTONS
        JMP* P.R                RETURN

PS      IOT 2                   DISABLE INTERRUPTS
        JMS PS1                 SET PUSH BUTTONS
        IOT 42                  ENABLE INTERRUPTS
        JMP* P.S                RETURN

PN      JMS P.E                 ENABLE MANUAL OPN OF PUSH BUTTONS
        JMS T.F                 TERMINATE TASK

PS1     SDC 0
        DAC PRG                 STORE NEW PUSH BUTTON STATUS
        LRS 6                   SHIFT BITS 0-5 INTO POSITION
        AND =77                 TRUNCATE HIGH ORDER BITS
        TAD =200                SET BITS 0-5 ENABLE BIT
        IOT 705                 SET PUSH BUTTONS 0-5
        LLS 6                   SHIFT BITS 6-11 INTO POSITION
        AND =77                 TRUNCATE HIGH ORDER BITS
        TAD =300                SET BITS 6-11 ENABLE BITS
        IOT 705                 SET PUSH BUTTONS 6-11
        JMP* PS1                RETURN

IPB     LAC PE+1                GET PUSH BUTTON ENABLE SWITCH
        SNA                     SKIP IF PUSH BUTTONS ARE ENABLED
        JMP IPB1                RESTORE PUSH BUTTON STATUS
```

```
        LAC PTT          GET ADDRESS OF PUSH BUTTON SERVICE
        AND =77777       TRUNCATE HIGH ORDER BITS
        JMS TII          SCHEDULE PUSH BUTTON SERVICE
        IOT 631          READ PUSH BUTTONS
        DAC PRG          MODIFY PUSH BUTTON STATUS WORD
        DZM PE+1         DISABLE PUSH BUTTONS
        JMP IR           RETURN FROM INTERRUPT
IPB1    LAC PRG          GET FORMER PUSH BUTTON STATUS
        JMS PSI          SET PUSH BUTTONS
        JMP IR           RETURN FROM INTERRUPT
```

```
        STITLE                  DISPLAY COMMUNICATOR

DE      JMS *+1                 SET DISPLAY INT ENABLE SWITCH
        SDC 0                   DISPLAY INT ENABLE SWITCH
        JMP* D.E                RETURN

DD      DZM DE+1                CLEAR INT ENABLE SWITCH
        JMP* D.D                RETURN

DP      SNA                     SKIP IF NOT NULL SERVICE
        LAC =DN                 GET ADDRESS OF NULL SERVICE
        DAC DPT                 STORE ADDRESS OF SERVICE TASK
        JMP* D.P                RETURN

DA      LAC DS1                 GET STATUS WORD 1
        LLS 14                  SHIFT BREAK FIELD INTO POSITION
        AND =70000              REMOVE ALL BUT BREAK FIELD
        XOR DSA                 FORM 15-BIT ADDRESS
        JMP* D.A                RETURN

DY      LAC DS2                 GET STATUS WORD 2
        LLS 3                   SHIFT HIGH ORDER BIT INTO POSITION
        AND =10000              REMOVE OTHER BITS
        XOR DSY                 FORM 13-BIT Y COORDINATE
        TAD =-1000              CONVERT RELATIVE TO SCREEN CENTER
        JMP* D.Y                RETURN

DX      LAC DS2                 GET STATUS WORD 2
        LLS 4                   SHIFT HIGH ORDER BIT INTO POSITION
        AND =10000              REMOVE OTHER BITS
        XOR DSX                 FORM 13-BIT X COORDINATE
        TAD =-1000              CONVERT RELATIVE TO SCREEN CENTER
        JMP* D.X                RETURN

DO      AND =77                 TRUNCATE HIGH ORDER BITS
        RCL                     MULTIPLY PARAMETER BY 2
        CMA                     FORM 1'S COMPLEMENT
        TAD DSP                 ADD PUSH DOWN POINTER
        TAD =PDP2-PDP1-1        COMPUTE ADDRESS OF PUSH DOWN ENTRY
        DAC T1                  SAVE TEMPORARILY
```

```
        TAD =-PDP2          FORM VALIDITY CHECK
        SPA                 SKIP IF PARAMETER VALID
        JMP* D.O            NOT ENOUGH OWNERS
        LAC* T1             GET FIRST PUSH DOWN WORD
        LLS 3               SHIFT BREAK FIELD INTO POSITION
        AND =70000          REMOVE ALL BUT BREAK FIELD
        ISZ T1              SET POINTER TO SECOND PD ENTRY
        TAD* T1             COMBINE FIRST & SECOND ENTRIES
        TAD =7777           FORM ADDRESS IN OWNER OF OWNER
        DAC T1              SAVE TEMPORARILY
        LAC* T1             GET ADDRESS OF DESIRED OWNER
        ISZ D.O             INDICATE SUCCESS
        JMP* D.O            RETURN

DN      JMS D.E             ENABLE DISPLAY INTERRUPTS
        JMS T.F             TERMINATE TASK

DW      $DC 0
        DZM DWT             CLEAR DISPLAY READY SWITCH
        LAC DWT             GET DISPLAY READY SWITCH
        SNA                 SKIP IF SET
        JMP *+3             WAIT FOR DISPLAY TO FINISH FRAME
        JMP* DW             RETURN
        JMP DW+2            CHECK DISPLAY READY SWITCH
        JMS T.P             SCHEDULE NEW SWITCH CHECK
DWT     $DC 0               DISPLAY READY SWITCH
        LAC DWV             GET TRANSLATION VALUE
        SNA                 SKIP IF TRANSLATION PENDING
        JMP XIS1            RESUME DISPLAY & RETURN
        DAC* DWHD           STORE DISPLACEMENT
        XOR =2000           INVERT SIGN BIT
        DAC* DWTL           STORE COUNTERDISPLACEMENT
        DZM DWV             INDICATE TRANSLATION PERFORMED
        JMP XIS1            RESUME DISPLAY & RETURN

ILP     LAC DSS             GET DISPLAY STATUS WORD 1
        AND =7              GET BREAK FIELD
        SZA                 SKIP IF ZERO BREAK FIELD
        JMP *+5             USER FILE INTERRUPT
        IOT 611             READ DISPLAY ADDRESS
```

```
        TAD =-XP            FORM ADDRESS CHECK
        SMA                 SKIP IF USER FILE INTERRUPT
        JMP XLP             TRACKING INTERRUPT
        LAC DE+1            GET DISPLAY INT ENABLE SWITCH
        SZA                 SKIP IF DISPLAY INTERRUPTS DISABLED
        JMP *+3             GET STATUS FOR USER
        IOT 724             RESUME DISPLAY
        JMP IR              RETURN FROM INTERRUPT
        LAC UPT             GET ADDRESS OF SERVICE TASK
        JMS DS              SCHEDULE SERVICE & READ STATUS
        JMP *-4             RESUME DISPLAY & RETURN

IIS     LAC DSS             GET DISPLAY STATUS WORD 1
        AND =7              GET BREAK FIELD
        SNA                 SKIP IF USER FILE INTERRUPT
        JMP XIS             TRACKING INTERRUPT
        IOT 611             READ DISPLAY ADDRESS
        XOR =10000          INTERPRET WITH BREAK FIELD 1
        SAD =D+4            SKIP IF NOT DISPLAY SYNC INTERRUPT
        JMS DWT             SET DISPLAY READY SWITCH
        LAC DE+1            GET DISPLAY INT ENABLE SWITCH
        SZA                 SKIP IF DISPLAY INTERRUPTS DISABLED
        JMP *+5             GET STATUS FOR USER
        IOT 611             READ DISPLAY ADDRESS
        TAD =1              FORM RESUME ADDRESS
        IOT 1605            RESUME DISPLAY
        JMP IR              RETURN FROM INTERRUPT
        LAC DSS             GET DISPLAY STATUS WORD 1
        LLS 14              SHIFT BREAK FIELD INTO POSITION
        AND =70000          REMOVE ALL BUT BREAK FIELD
        IOT 601             FORM DISPLAY ADDRESS
        DAC 23              SAVE TEMPORARILY
        LAC* 23             GET ADDRESS OF SERVICE TASK
        JMS DS              SCHEDULE SERVICE & READ STATUS
        JMP *-13            RESUME DISPLAY & RETURN

DS      SDC 0
        AND =77777          TRUNCATE HIGH ORDER BITS
        JMS TII             SCHEDULE SERVICE TASK
        DZM DE+1            DISABLE DISPLAY INTERRUPTS
```

```
LAC DSS             GET DISPLAY STATUS WORD 1
DAC DS1             SAVE
IOT 1632            READ STATUS WORD 2
DAC DS2             SAVE
IOT 611             READ DISPLAY ADDRESS
DAC DSA             SAVE
IOT 1612            READ Y DISPLAY COORDINATE
DAC DSY             SAVE
IOT 512             READ X DISPLAY COORDINATE
DAC DSX             SAVE
IOT 511             READ PUSH DOWN POINTER
DAC DSP             SAVE
LAC =PDP1-1         GET ADDRESS OF PUSH DOWN LIST
DAC 10              SET AUTOINDEX REGISTER
LAC =PDP2-1         GET ADDRESS OF PUSH DOWN SAVE AREA
DAC 11              SET AUTOINDEX REGISTER
LAC* 10             GET WORD FROM PUSH DOWN LIST
DAC* 11             STORE IN PUSH-DOWN SAVE AREA
LAC 10              GET SOURCE POINTER
SAD DSP             SKIP IF NOT END OF LIST
JMP* DS             RETURN
JMP *-5             COPY NEXT WORD
```

```
        STITLE                  TRACKING CONTROLLER

XI      TAD =1000               CONVERT RELATIVE TO ORIGIN
        AND =1777               CONVERT MODULO 2↑10
        DAC XPY                 SET Y TRACKING COORDINATE
        LACQ                    GET X COORDINATE
        TAD =1000               CONVERT RELATIVE TO ORIGIN
        AND =1777               CONVERT MODULO 2↑10
        XOR =4000               SET ESCAPE BIT
        DAC XPX                 SET X TRACKING COORDINATE
        DZM XP                  ENABLE TRACKING
        JMP* X.I                RETURN

XR      DZM XP                  ENABLE TRACKING
        JMP* X.R                RETURN

XT      LAW 3000                GET POP INSTRUCTION
        DAC XP                  TERMINATE TRACKING
        JMP* X.T                RETURN

XS      LAW 3000                GET POP INSTRUCTION
        SAD XP                  SKIP IF TRACKING ENABLED
        ISZ X.S                 INDICATE SUCCESS
        JMP* X.S                RETURN

XY      LAC XPY                 GET Y TRACKING COORDINATE
        TAD =-1000              CONVERT RELATIVE TO SCREEN CENTER
        JMP* X.Y                RETURN

XX      LAC XPX                 GET X TRACKING COORDINATE
        AND =1777               TRUNCATE ESCAPE
        TAD =-1000              CONVERT RELATIVE TO SCREEN CENTER
        JMP* X.X                RETURN

XLP     HLT                     STATE VARIABLE
        HLT                     STATE VARIABLE
        HLT                     STATE VARIABLE
        JMP *+3                 DO NOT CHANGE Y TRACKING COORDINATE
        AND =1777               TRUNCATE HIGH ORDER BITS
        DAC XPY                 SET Y TRACKING COORDINATE
```

```
        IOT 512             READ X COORDINATE
        TAD =-2000          SUBTRACT 1024
        SMA                 SKIP IF COORDINATE ON SCREEN
        JMP *+4             DO NOT CHANGE X COORDINATE
        AND =1777           TRUNCATE HIGH ORDER BITS
        XOR =4000           SET ESCAPE BIT
        DAC XPX             SET X TRACKING COORDINATE
        LAW XP              GET ADDRESS OF TRACKING PATTERN
        IOT 1605            RESTART TRACKING PROCESS
        JMP IR              RETURN FROM INTERRUPT
XLP1    IOT 724             RESUME DISPLAY
        JMP IR              RETURN

XIS     IOT 611             READ DISPLAY ADDRESS
        DAC 23              SAVE TEMPORARILY
        LAC* 23             GET ADDRESS OF SERVICE
        DAC 23              SAVE TEMPORARILY
        JMP* 23             SERVICE INTERRUPT
XIS1    IOT 611             READ DISPLAY ADDRESS
        TAD =1              FORM RESUME ADDRESS
        IOT 1605            RESUME DISPLAY
        JMP IR              RETURN FROM INTERRUPT

X1      LAW 3000            GET POP INSTRUCTION
        DAC XPS             INHIBIT SEARCH PATTERN
        LAC =700512         GET IOT 512 INSTRUCTION
        DAC XLP             MODIFY INTERRUPT SERVICE
        LAC =40000+XL       GET DAC XL INSTRUCTION
        DAC XLP+1           MODIFY INTERRUPT SERVICE
        LAC =600000+XLP1    GET JMP XLP1 INSTRUCTION
        DAC XLP+2           MODIFY INTERRUPT SERVICE
        DZM XL              CLEAR LOW COORDINATE
        DZM XH              CLEAR HIGH COORDINATE
        JMP XIS1            RESUME DISPLAY & RETURN

X2      LAC =40000+XH       GET DAC XH INSTRUCTION
        DAC XLP+1           MODIFY INTERRUPT SERVICE
        JMP XIS1            RESUME DISPLAY & RETURN

X3      LAC XH              GET HIGH COORDINATE
```

```
        SNA                 SKIP IF VALID
        JMP X31             ENABLE SEARCH PATTERN
        TAD XL              ADD LOW COORDINATE
        SAD XH              SKIP IF VALID
        JMP X31             ENABLE SEARCH PATTERN
        RCR                 DIVIDE BY 2
        TAD =-2000          SUBTRACT 1024
        SMA                 SKIP IF COORDINATE ON SCREEN
        JMP X31+2           DO NOT CHANGE X COORDINATE
        AND =1777           CONVERT MODULO 2↑10
        XOR =4000           SET ESCAPE BIT
        DAC XPX             SET X TRACKING COORDINATE
        JMP X31+2           LEAVE SEARCH PATTERN INHIBITED
X31     LAW 777             GET SEARCH ENABLE WORD
        DAC XPS             ENABLE SEARCH PATTERN
        DZM XL              CLEAR LOW COORDINATE
        DZM XH              CLEAR HIGH COORDINATE
        LAC =701612         GET IOT 1612 INSTRUCTION
        DAC XLP             MODIFY INTERRUPT SERVICE
        LAC =40000+XL       GET DAC XL INSTRUCTION
        DAC XLP+1           MODIFY INTERRUPT SERVICE
        JMP XIS1            RESUME DISPLAY & RETURN

X4      LAC XH              GET HIGH COORDINATE
        SNA                 SKIP IF NOT VALID
        JMP X41             ENABLE SEARCH PATTERN
        TAD XL              ADD LOW COORDINATE
        SAD XH              SKIP IF VALID
        JMP X41             ENABLE SEARCH PATTERN
        RCR                 DIVIDE BY 2
        TAD =-2000          SUBTRACT 1024
        SMA                 SKIP IF COORDINATE ON SCREEN
        JMP X41+2           DO NOT CHANGE Y TRACKING COORDINATE
        AND =1777           CONVERT MODULO 2↑10
        DAC XPY             SET Y TRACKING COORDINATE
        JMP X41+2           LEAVE SEARCH PATTERN INHIBITED
X41     LAW 777             GET SEARCH ENABLE WORD
        DAC XPS             ENABLE SEARCH PATTERN
        LAC XLP+7           GET TAD =-2000 INSTRUCTION
        DAC XLP+1           MODIFY INTERRUPT SERVICE
```

```
        LAC =740100         GET SMA INSTRUCTION
        DAC XLP+2           MODIFY INTERRUPT SERVICE
        JMP XIS1            RESUME DISPLAY & RETURN

X5      LAW 3000            GET POP INSTRUCTION
        DAC XP              DISABLE TRACKING
        JMP XIS1            RESUME DISPLAY & RETURN
```

```
        STITLE          STRUCTURE TOPOLOGY OPERATORS

STL     JMS B4          GET 8-WORD BLOCK
        JMP* S.TL       NOT ENOUGH STORAGE
        DAC T5          SAVE ADDRESS FOR RETURN
        TAD =-1         COMPUTE INITIAL INDEX VALUE
        DAC 12          SET AUTOINDEX REGISTER
        LAW 0           GET DISPLAY NOP INSTRUCTION
        DAC* 12         STORE IN FIRST LOCATION IN HEAD
        DAC* 12         STORE IN SECOND LOCATION IN HEAD
        DAC* 12         STORE IN THIRD LOCATION IN HEAD
        LAW 1121        GET VEC INSTRUCTION
        DAC* 12         STORE IN FOURTH LOCATION IN HEAD
        LAW 0           GET DISPLAY NOP INSTRUCTION
        DAC* 12         STORE IN FIFTH LOCATION IN HEAD
        LAW 4000        GET ZERO X COORD WITH ESCAPE BIT
        DAC* 12         STORE IN SIXTH LOCATION IN HEAD
        LAW 2001        GET JUMP1 INSTRUCTION
        DAC* 12         STORE IN SEVENTH LOCATION IN HEAD
        JMS B4          GET 8-WORD BLOCK
        JMP STL1        NOT ENOUGH STORAGE
        DAC* 12         STORE ADDRESS OF TAIL IN HEAD
        TAD =-1         COMPUTE INITIAL INDEX VALUE
        DAC 12          SET AUTOINDEX REGISTER
        LAW 6240        GET UNCONDITIONAL DISPLAY SKIP
        DAC* 12         STORE IN FIRST LOCATION IN TAIL
        LAW 1400        GET INTERNAL STOP INSTRUCTION
        DAC* 12         STORE IN SECOND LOCATION IN TAIL
        DZM* 12         ZERO IN THIRD LOCATION IN TAIL
        DZM* 12         STORE DISPLAY NOP IN FOURTH LOCATION
        LAW 1121        GET VEC INSTRUCTION
        DAC* 12         STORE IN FIFTH LOC IN TAIL
        DZM* 12         STORE IN SIXTH LOC IN TAIL
        LAW 4000        GET ZERO X COORD WITH ESCAPE BIT
        DAC* 12         STORE IN SEVENTH LOCATION IN TAIL
        LAW 3000        GET POP INSTRUCTION
        DAC* 12         STORE IN EIGHTH LOC IN TAIL
        LAC T5          GET ADDRESS OF CREATED LEVEL
        ISZ S.TL        INDICATE SUCCESS
        JMP* S.TL       RETURN
```

```
STL1    DZM* T5          FREE FIRST 4-WORD BLOCK IN HEAD
        LAC  ,           GET ADDRESS OF 8-WORD BLOCK
        TAD  -           FORM ADDRESS OF SECOND 4-WORD BLOCK
        DAC T5           SAVE TEMPORARILY
        DZM* T5          FREE SECOND 4-WORD BLOCK IN HEAD
        JMP* S.TL        FAILURE RETURN

STD     SAD =DHAL        SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP STD1         HIGHEST ACTIVE LEVEL
        DAC T1           SAVE ADDRESS OF FIRST HEAD BLOCK
        TAD =4           FORM ADDRESS OF SECOND HEAD BLOCK
        DAC T2           SAVE ADDRESS OF SECOND HEAD BLOCK
        TAD =3           FORM POINTER TO LAST LOC IN HEAD
        DAC T3           SAVE TEMPORARILY
        LAC* T3          GET ADDRESS OF TAIL (OR NODE)
        DAC T3           SAVE ADDRESS OF TAIL (OR NODE)
        TAD =4           FORM ADDRESS OF SECOND TAIL BLOCK
        DAC T4           SAVE
        LAC* T3          GET FIRST WORD OF TAIL (OR NODE)
        AND =777770      TRUNCATE BREAK FIELD
        SAD =762010      SKIP IF NOT NODE
        JMP* S.TD        LEVEL NOT EMPTY
        DZM* T1          RELEASE FIRST HEAD BLOCK
        DZM* T2          RELEASE SECOND HEAD BLOCK
        DZM* T3          RELEASE FIRST TAIL BLOCK
        DZM* T4          RELEASE SECOND TAIL BLOCK
STD1    ISZ S.TD         INDICATE SUCCESS
        JMP* S.TD        RETURN

STI     TAD =7           FORM POINTER TO LAST LOC IN HEAD
        DAC T5           SAVE
        JMS B3           CREATE 4-WORD BLOCK
        JMP* S.TI        NOT ENOUGH STORAGE
        DAC T1           SET POINTER TO BLOCK
        TAD =-1          COMPUTE INITIAL INDEX VALUE
        DAC 12           SET AUTOINDEX REGISTER
        LLS 6            SHIFT BREAK FIELD INTO AC
        AND =7           TRUNCATE HIGH ORDER BITS
        XOR =762010      FORM PUSH JUMP INSTRUCTION
        DAC* 12          STORE IN FIRST LOC IN BLOCK
```

```
        AND =7              TRUNCATE HIGH ORDER BITS
        LLS 14              GET COMPLETE ADDRESS
        DAC* 12             STORE IN SECOND LOC IN BLOCK
        LAW 2001            GET JUMP1 INSTRUCTION
        DAC* 12             STORE IN THIRD LOC IN BLOCK
        LAC* T5             GET ADR OF FIRST ELEMENT IN LEVEL
        DAC* 12             STORE AS SUCCESSOR TO NEW NODE
        LAC T1              GET ADDRESS OF NEW NODE
        DAC* T5             INSERT NEW NODE INTO LEVEL
        ISZ S.TI            INDICATE SUCCESS
        JMP* S.TI           RETURN

STR     JMS TV              PROTECT AGAINST REENTRY
        $DC S.TR
        $DC 0
        TAD =7              GET POINTER TO END OF HEAD
        DAC T1              SAVE TEMPORARILY
        LAC* T1             GET ADDRESS OF FIRST ELEMENT
        DAC T2              SAVE TEMPORARILY
        DAC STR2            SAVE ADDRESS FOR REMOVAL
        LAC* T2             GET FIRST WORD OF FIRST ELEMENT
        AND =777770         TRUNCATE BREAK FIELD
        SAD =762010         SKIP IF NOT NODE
        SKP                 NODE
        JMP STR1+7          SUBSTRUCTURE NOT IN LEVEL
        ISZ T2              FORM POINTER TO ADR OF SUBSTRUCTURE
        LACQ                GET ADDRESS OF GIVEN SUBSTRUCTURE
        SAD* T2             SKIP IF NO MATCH
        JMP STR1            SUBSTRUCTURE FOUND
        LAC T2              GET POINTER TO ADR OF SUBSTRUCTURE
        TAD =2              FORM POINTER TO END OF NODE
        JMP STR+4           TRY NEXT NODE
STR1    ISZ T2              INCREMENT POINTER TO LOC IN NODE
        ISZ T2              INCREMENT POINTER TO LOC IN NODE
        LAC* T2             GET ADR OF SUCCESSOR TO NODE
        DAC* T1             STORE IN PREVIOUS NODE (OR HEAD)
        JMS DW              WAIT FOR DISPLAY TO SETTLE DOWN
        DZM* STR2           RELEASE NODE TO FREE STORAGE
        ISZ STR+2           INDICATE SUCCESS
        JMS T.U             UNLOCK S.TRD
```

SDC STR

```
        STITLE                  LEVEL MODIFICATION OPERATORS

SLH     LAC =DHAL               GET ADDRESS OF HIGHEST ACTIVE LEVEL
        JMP* S.LH               RETURN

SLY     JMS TV                  PROTECT AGAINST REENTRY
        $DC S.LY
        $DC 0
        SAD =DHAL               SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP SLY1+3              RETURN
        TAD =4                  FORM POINTER TO Y COORD IN HEAD
        DAC DWHD                SAVE
        TAD =3                  FORM POINTER TO END OF HEAD
        JMS SLT                 GET ADDRESS OF TAIL
        TAD =5                  FORM POINTER TO Y COORD IN TAIL
        DAC DWTL                SAVE
        LACQ                    GET Y INCREMENT
        JMS C.BC                CONVERT TO DISPLAY COORDINATE
        XOR =760000             INDICATE STORAGE OCCUPIED
        DAC DWV                 SAVE TRANSLATION VALUE
SLY1    LAC DWV                 GET TRANSLATION VALUE
        SZA                     SKIP IF TRANSLATION COMPLETE
        JMP *+4                 RESCHEDULE COMPLETION CHECK
        JMS T.U                 UNLOCK S.LY
        $DC SLY
        JMP SLY1                CHECK FOR TRANSLATION COMPLETE
        JMS T.P                 SCHEDULE COMPLETION CHECK

SLX     JMS TV                  PROTECT AGAINST REENTRY
        $DC S.LX
        $DC 0
        SAD =DHAL               SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP SLX1+3              RETURN
        TAD =5                  FORM POINTER TO X COORD IN HEAD
        DAC DWHD                SAVE
        TAD =2                  FORM POINTER TO END OF HEAD
        JMS SLT                 GET ADDRESS OF TAIL
        TAD =6                  FORM POINTER TO X COORE IN TAIL
        DAC DWTL                SAVE
        LACQ                    GET X INCREMENT
```

```
        JMS C.BC            CONVERT TO DISPLAY COORDINATE
        XOR =4000           SET ESCAPE BIT
        DAC DWV             SAVE TRANSLATION VALUE
SLX1    LAC DWV             GET TRANSLATION VALUE
        SZA                 SKIP IF TRANSLATION COMPLETE
        JMP .+4             RESCHEDULE COMPLETION CHECK
        JMS T.U             UNLOCK S.LX
        SDC SLX
        JMP SLX1            CHECK FOR TRANSLATION COMPLETE
        JMS T.P             SCHEDULE COMPLETION CHECK

SLP     SAD =DHAL           SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP* S.LP           RETURN
        TAD =2              GET ADDRESS OF PARAMETER SLOT
        DAC T1              SAVE TEMPORARILY
        LACQ                GET PARAMETERS
        AND =777            TRUNCATE HIGH ORDER BITS
        DAC* T1             STORE PARAMETERS IN LEVEL
        JMP* S.LP           RETURN

SLBE    SAD =DHAL           SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP* S.LBE          RETURN
        TAD =1              FORM POINTER TO COUNT SLOT
        DAC T2              SAVE TEMPORARILY
        TAD =6              FORM POINTER TO END OF HEAD
        JMS SLT             GET ADDRESS OF TAIL
        TAD =3              FORM POINTER TO BLINK OFF SLOT
        DAC T1              SAVE TEMPORARILY
        LAW 6301            GET BLINK OFF INSTRUCTION
        DAC* T1             STORE IN TAIL
        LAC* T2             GET COUNT INSTRUCTION
        AND =74             GET COUNT BITS
        TAD =6302           FORM NEW COUNT INSTRUCTION
        DAC* T2             STORE NEW COUNT INSTRUCTION
        JMP* S.LBE          RETURN

SLBD    SAD =DHAL           SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP* S.LBD          RETURN
        TAD =1              FORM POINTER TO COUNT SLOT
        DAC T2              SAVE TEMPORARILY
```

```
        TAD =6          FORM POINTER TO END OF HEAD
        JMS SLT         GET ADDRESS OF TAIL
        TAD =3          FORM POINTER TO BLINK OFF SLOT
        DAC T1          SAVE TEMPORARILY
        LAC* T2         GET COUNT INSTRUCTION
        AND =777774     FORM NEW COUNT INSTRUCTION
        DAC* T2         STORE NEW COUNT INSTRUCTION
        DZM* T1         REMOVE BLINK OFF INSTRUCTION
        JMP* S.LBD      RETURN

SLC     SAD =DHAL       SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP* S.LC       RETURN
        TAD =1          FORM POINTER TO COUNT SLOT
        DAC T1          SAVE TEMPORARILY
        LAC* T1         GET COUNT INSTRUCTION
        AND =2          GET BLINK BIT
        DAC T2          SAVE TEMPORARILY
        LACQ            GET COUNT BITS
        AND =7A         TRUNCATE OTHER BITS
        XOR T2          CONCATENATE COUNT BITS & BLINK BIT
        XOR =6300       FORM NEW COUNT INSTRUCTION
        DAC* T1         STORE NEW COUNT INSTRUCTION
        JMP* S.LC       RETURN

SLU     SAD =DHAL       SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP* S.LU       RETURN
        JMS S.LN        REMOVE INTERRUPT AT END OF LEVEL
        DAC* T1         REMOVE SKIP INSTRUCTION FROM TAIL
        JMP* S.LU       RETURN

SLS     SAD =DHAL       SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP* S.LS       RETURN
        JMS S.LN        REMOVE INTERRUPT AT END OF LEVEL
        LAW 6220        GET SKIP-IF-OFF-SCREEN INSTRUCTION
        DAC* T1         STORE IN TAIL
        JMP* S.LS       RETURN

SLL     SAD =DHAL       SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP* S.LL       RETURN
        JMS SLSP        REMOVE INTERRUPT FROM END OF LEVEL
```

```
        LAW 6201            GET LPSI CLEAR INSTRUCTION
        DAC* T2             STORE IN HEAD
        LAW 6202            GET SKIP-ON-NO-LPSI INSTRUCTION
        DAC* T1             STORE IN TAIL
        JMP* S.LL           RETURN

SLN     SAD =DHAL           SKIP IF NOT HIGHEST ACTIVE LEVEL
        JMP* S.LN           RETURN
        JMS SLSP            REMOVE INTERRUPT AT END OF LEVEL
        LAW 0               GET DISPLAY NOP INSTRUCTION
        DAC* T2             REMOVE LPSI CLEAR
        JMP* S.LN           RETURN

SLT     $DC 0
        DAC T1              STORE POINTER TO END OF BLOCK
        LAC* T1             GET POINTER TO NEXT NODE (OR TAIL)
        DAC T1              SAVE TEMPORARILY
        LAC* T1             GET FIRST WORD FROM NODE (OR TAIL)
        AND =777770         TRUNCATE BREAK FIELD
        SAD =762010         SKIP IF NOT NODE
        JMP *+3             TAIL NOT FOUND
        LAC T1              GET ADDRESS OF TAIL
        JMP* SLT            RETURN
        LAC T1              GET POINTER TO NODE
        TAD =3              FORM POINTER TO END OF NODE
        JMP SLT+1           LOOK AT NEXT NODE (OR TAIL)

SLSP    $DC 0
        DAC T2              SAVE ADDRESS OF LEVEL
        TAD =7              FORM POINTER TO END OF HEAD
        JMS SLT             GET ADDRESS OF TAIL
        TAD =2              FORM POINTER TO TASK ADDRESS
        DAC T3              SAVE TEMPORARILY
        LAW 6240            GET SKIP INSTRUCTION
        DAC* T1             STORE IN TAIL
        LACQ                GET NEW SERVICE TASK ADDRESS
        DAC* T3             STORE IN TAIL
        JMP* SLSP           RETURN
```

```
        $TITLE              TEXT OPERATORS

LT      DAC LT2             SAVE ADDRESS OF TEXT LIST
        LAC* LT2            GET TEXT WORD COUNT
        CMA                 FORM 1'S COMPLEMENT
        DAC LT3             STORE COMPLEMENTED WORD COUNT
        ISZ LT3             FORM 2'S COMPLEMENT OF WORD COUNT
        SKP                 WORD COUNT NOT ZERO
        JMP* L.T            RETURN
LT1     ISZ LT2             SET POINTER TO NEXT TEXT WORD
        LAC* LT2            GET TEXT WORD
        JMS B.T             SEND TO TELEPRINTER BUFFER
        ISZ LT3             INCREMENT COUNT & SKIP IF DONE
        JMP LT1             PROCESS NEXT TEXT WORD
        JMP* L.T            RETURN

LD      DAC T5              SAVE ADDRESS OF TEXT LIST
        DAC T1              SET POINTER TO TEXT LIST
        LAC* T1             GET WORD COUNT
        CMA                 FORM 1'S COMPLEMENT
        DAC T2              STORE COMPLEMENTED WORD COUNT
        LAC =7              GET INITIAL VALUE OF LEAF LENGTH
        DAC T3              SET INITIAL VALUE OF LEAF LENGTH
        ISZ T2              FORM 2'S COMPLEMENT OF WORD COUNT
        SKP                 WORD COUNT NOT ZERO
        JMP LD4             RETURN NULL TEXT LEAF
LD1     ISZ T1              SET POINTER TO NEXT TEXT WORD
        LAC* T1             GET TEXT WORD
        LRS 14              SHIFT FIRST CHARACTER INTO POSITION
        JMS LD5             MODIFY LEAF LENGTH COUNT
        JMS LD5             MODIFY LEAF LENGTH COUNT
        JMS LD5             MODIFY LEAF LENGTH COUNT
        ISZ T2              INCREMENT WORD COUNT & SKIP IF DONE
        JMP LD1             PROCESS NEXT TEXT WORD
        LAC T3              GET SIZE OF TEXT LEAF
        RCR                 DIVIDE BY 2
        RCR                 DIVIDE BY 2
        JMS B               GET STORAGE FOR TEXT LEAF
        JMP* L.D            NOT ENOUGH STORAGE
        DAC T1              SAVE ADDRESS OF TEXT LEAF AREA
```

```
        DAC T6              SAVE ADDRESS FOR RETURN
        LAC* T5             GET WORD COUNT
        CMA                 FORM 1'S COMPLEMENT
        TAD =1              FORM 2'S COMPLEMENT
        DAC T2              STORE COMPLEMENTED WORD COUNT
        DZM T3              CLEAR HORIZONTAL COUNT
        DZM T4              CLEAR VERTICAL COUNT
LD2     ISZ T5              SET POINTER TO NEXT TEXT WORD
        LAC* T5             GET NEXT TEXT WORD
        LRS 14              SHIFT FIRST CHARACTER INTO POSITION
        JMS LD6             PUT FIRST CHARACTER INTO LEAF
        JMS LD6             PUT SECOND CHARACTER INTO LEAF
        JMS LD6             PUT THIRD CHARACTER INTO LEAF
        ISZ T2              INCREMENT COUNT & SKIP IF DONE
        JMP LD2             PROCESS NEXT TEXT WORD
        LAW 1121            GET VEC INSTRUCTION
        DAC* T1             STORE IN TEXT LEAF
        ISZ T1              INCREMENT POINTER TO LOC IN LEAF
        LAC T4              GET VERTICAL COUNT
        CLQ                 PREPARE TO SHIFT ZEROS INTO AC
        LLS 4               MULTIPLY BY 16
        XOR =400000         SET TO NONZERO VALUE
        DAC* T1             STORE IN TEXT LEAF
        ISZ T1              INCREMENT POINTER TO LOC IN LEAF
        LAC T3              GET HORIZONTAL COUNT
        LLS 3               MULTIPLY BY 8
        AND =1777           CONVERT MODULO 2↑10
        XOR =6000           SET ESCAPE BIT & MINUS SIGN
        DAC* T1             STORE IN TEXT LEAF
        ISZ T1              INCREMENT POINTER TO LOC IN LEAF
LD3     LAW 3000            GET POP INSTRUCTION
        DAC* T1             STORE IN TEXT LEAF
        LAC T6              GET ADDRESS OF TEXT LEAF
        ISZ L.D             INDICATE SUCCESS
        JMP* L.D            RETURN
LD4     LAC =LD3            GET ADDRESS OF POP INSTRUCTION
        JMP *-3             INDICATE SUCCESS & RETURN

LD5     SDC 0
        AND =77             TRUNCATE HIGH ORDER BITS
```

```
        SAD =77          SKIP IF NOT NULL CHARACTER
        JMP *+5          NULL CHARACTER -- RETURN
        SAD =74          SKIP IF NOT CARRIAGE RETURN
        ISZ T3           INCREMENT LEAF SIZE EXTRA TIME
        ISZ T3           INCREMENT LEAF SIZE
        ISZ T3           INCREMENT LEAF SIZE
        LLS 6            SHIFT NEXT CHARACTER INTO POSITION
        JMP* LD5         RETURN

LD6     SDC 0
        AND =77          TRUNCATE HIGH ORDER BITS
        SAD =77          SKIP IF NOT NULL CHARACTER
        JMP LD7-2        NULL CHARACTER -- RETURN
        SAD =74          SKIP IF NOT CARRIAGE RETURN
        JMP LD7          PUT CARRIAGE RETURN INTO LEAF
        SAD =75          SKIP IF NOT LINE FEED
        SKP              LINE FEED -- INCREMENT VERT COUNT
        JMP *+3          NORMAL CHARACTER
        ISZ T4           INCREMENT VERTICAL COUNT
        SKP              LEAVE HORIZONTAL COUNT ALONE
        ISZ T3           INCREMENT HORIZONTAL COUNT
        TAD =LD8         ADD ADDRESS OF CONVERSION TABLE
        DAC T7           SAVE TEMPORARILY
        LAW 2010         GET PUSH JUMP INSTRUCTION
        DAC* T1          STORE IN TEXT LEAF
        ISZ T1           INCREMENT POINTER TO LOC IN LEAF
        LAC* T7          GET ADDRESS OF DISPLAY FOR CHAR
        DAC* T1          STORE IN TEXT LEAF
        ISZ T1           INCREMENT POINTER TO LOC IN LEAF
        LLS 6            SHIFT NEXT CHARACTER INTO POSITION
        JMP* LD6         RETURN
LD7     LACQ             GET MQ CONTENTS
        DAC T7           SAVE TEMPORARILY
        LAW 1121         GET VEC INSTRUCTION
        DAC* T1          STORE IN TEXT LEAF
        ISZ T1           INCREMENT POINTER TO LOC IN LEAF
        LAW 0            GET ZERO Y DISPLACEMENT
        DAC* T1          STORE ZERO Y DISPLACEMENT IN LEAF
        ISZ T1           INCREMENT POINTER TO LOC IN LEAF
        LAC T3           GET HORIZONTAL DISPLACEMENT
```

```
        CLQ                     PREPARE TO SHIFT ZEROS INTO AC
        LLS 3                   MULTIPLY BY 8
        AND =1777               CONVERT MODULO 2↑10
        XOR =6000               SET ESCAPE BIT & MINUS SIGN
        DAC* T1                 STORE IN TEXT LEAF
        ISZ T1                  INCREMENT POINTER TO LOC IN LEAF
        DZM T3                  CLEAR HORIZONTAL COUNT
        LAC T7                  GET PREVIOUS MQ CONTENTS
        LRS 14                  SHIFT NEXT CHARACTER INTO POSITION
        JMP* LD6                RETURN

LL      DAC T1                  STORE ADDRESS OF TEXT LEAF
        LAC* T1                 GET VALUE FROM LEAF
        DZM* T1                 FREE STORAGE LOCATION
        SAD =763000             SKIP IF NOT END OF TEXT LEAF
        JMP* L.L                RETURN
        ISZ T1                  SET POINTER TO NEXT LOC IN LEAF
        JMP LL+1                FREE NEXT LOCATION

LD8     SDC D00
        SDC D01+10000
        SDC D02+20000
        SDC D03+30000
        SDC D04+40000
        SDC D05+50000
        SDC D06+60000
        SDC D07+70000
        SDC D10+100000
        SDC D11+110000
        SDC D12+120000
        SDC D13+130000
        SDC D14+140000
        SDC D15+150000
        SDC D16+160000
        SDC D17+170000
        SDC D20+200000
        SDC D21+210000
        SDC D22+220000
        SDC D23+230000
        SDC D24+240000
```

```
$DC  D25+250000
$DC  D26+260000
$DC  D27+270000
$DC  D30+300000
$DC  D31+310000
$DC  D32+320000
$DC  D33+330000
$DC  D34+340000
$DC  D35+350000
$DC  D36+360000
$DC  D37+370000
$DC  D40+400000
$DC  D41+410000
$DC  D42+420000
$DC  D43+430000
$DC  D44+440000
$DC  D45+450000
$DC  D46+460000
$DC  D47+470000
$DC  D50+500000
$DC  D51+510000
$DC  D52+520000
$DC  D53+530000
$DC  D54+540000
$DC  D55+550000
$DC  D56+560000
$DC  D57+570000
$DC  D60+600000
$DC  D61+610000
$DC  D62+620000
$DC  D63+630000
$DC  D64+640000
$DC  D65+650000
$DC  D66+660000
$DC  D67+670000
$DC  D70+700000
$DC  D71+710000
$DC  D72+720000
$DC  D73+730000
NOP
```

$DC D75+750000
$DC D76+760000

```
        STITLE              IDLE-TIME TASK

IDLE    LAW 17475           GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T             TYPE CARRIAGE RETURN, LINE FEED
        LAW 10              GET TELEPRINTER ALLOCATION MASK
        JMS T.R             RELEASE TELEPRINTER
        JMS B.K             GET KEYBOARD CHARACTER
        DAC T3              SAVE KEYBOARD CHARACTER
        LAW 10              GET TELEPRINTER ALLOCATION MASK
        JMS T.A             ALLOCATE TELEPRINTER
        LAC BKF             GET ASCII FORM OF CHARACTER
        SAD =205            SKIP IF NOT ENQUIRY
        JMP TTY4            SEND ENQUIRY RECORD
        LAC T3              GET KEYBOARD CHARACTER
        SAD =14             SKIP IF NOT C
        LAC =IDLEC          GET "CLEAR" RESPONSE POINTER
        SAD =33             SKIP IF NOT R
        LAC =IDLER          GET "RUN" RESPONSE POINTER
        SAD =34             SKIP IF NOT S
        LAC =IDLES          GET "SCHEDULE" RESPONSE POINTER
        SAD =72             SKIP IF NOT #
        LAC =IDLE1          GET TTY/201 RESPONSE POINTER
        SAD =17             SKIP IF NOT F
        LAC =IDLEF          GET "FROM" RESPONSE POINTER
        DAC T3              SAVE SELECTED RESPONSE POINTER
        AND =777700         TRUNCATE LOW ORDER BITS
        SNA                 SKIP IF LEGAL COMMAND
        JMP IDLEQ           CANCEL COMMAND
        LAC T3              GET RESPONSE POINTER
        TAD =1              COMPUTE ADDRESS OF TEXT LIST
        JMS L.T             TYPE TEXT LIST
        LAC* T3             GET ADDRESS OF RESPONSE
        DAC T3              SAVE TEMPORARILY
        JMP* T3             EXECUTE RESPONSE

IDLEQ   LAC =657475         GET QUESTION MARK CODE
        JMP IDLE+1          TYPE & GET NEW COMMAND

IDLEC   $DC CLEAR
        $DC 2
```

```
        $TEXT "CLEAR "
IDLER   $DC RUN
        $DC 2
        $TEXT "RUN"
        $DC 747575
IDLES   $DC SCHED
        $DC 3
        $TEXT "SCHEDULE "
IDLE1   $DC TTY201
        $DC 0
IDLEF   $DC FROM
        $DC 2
        $TEXT "FROM "

CLEAR   JMS B.K                 GET KEYBOARD CHARACTER
        SAD =15                 SKIP IF NOT D
        JMP CLEAR1              CLEAR DISPLAY STORAGE
        SAD =35                 SKIP IF NOT T
        SKP                     CLEAR TASK QUEUE
        JMP IDLEQ               CANCEL COMMAND
        LAC =CLEART             GET ADDRESS OF TEXT LIST
        JMS L.T                 TYPE TEXT LIST
        LAC =SCHEDQ             GET ADDRESS OF FROZEN TASK QUEUE
        JMS Q.C                 CLEAR FROZEN TASK QUEUE
        JMP IDLE                GET NEW COMMAND
CLEAR1  LAC =CLEARD             GET ADDRESS OF TEXT LIST
        JMS L.T                 TYPE TEXT LIST
        JMS STC                 CLEAR DISPLAY STORAGE
        DZM 25                  INDICATE NO DIAGNOSTIC
        JMP E1                  RE-ESTABLISH DISPLAYED TITLE

CLEARD  $DC 5
        $TEXT "DISPLAY STORAGE"

CLEART  $DC 4
        $TEXT "TASK QUEUE"

RUN     LAW 10                  GET TELEPRINTER MASK
        JMS T.R                 RELEASE TELEPRINTER
        JMS STC                 CLEAR DISPLAY STORAGE
```

```
RUN1    LAC =SCHEDQ             GET ADDRESS OF FROZEN TASK QUEUE
        JMS Q.F                 GET TASK FROM FROZEN TASK QUEUE
        JMS T.F                 TERMINATE IDLE-TIME EXECUTION
        JMS T.S                 SCHEDULE TASK FROM FROZEN QUEUE
        JMP RUN1                ENABLE NEXT TASK

SCHED   JMS OCTAL5              GET ADDRESS FROM KEYBOARD
        JMP IDLEQ               CANCEL COMMAND
        LMQ                     SET UP PARAMETER
        LAC =SCHEDQ             GET ADDRESS OF FROZEN TASK QUEUE
        JMS Q.A                 ADD TASK TO FROZEN QUEUE
        SKP                     TYPE DIAGNOSTIC
        JMP IDLE                GET NEW COMMAND
        LAC =SCHED1             GET ADDRESS OF TEXT LIST
        JMS L.T                 TYPE TEXT LIST
        JMP IDLE                GET NEW COMMAND

SCHED1  $DC 11
        $TEXT " -- NO ROOM FOR THIS TASK"

SCHEDQ  $DC *+37
        $DC *+36
        $DC *+35
        $ORG *+35

TTY201  LAW 17772               GET # CODE
        JMS B.T                 TYPE #
        JMS ECHO                ECHO KEYBOARD CHARACTER
        LAC BKF                 GET ASCII FORM OF CHARACTER
        SAD =215                SKIP IF NOT CARRIAGE RETURN
        JMP TTY1                TERMINATE RECORD WITH ETX
        SAD =337                SKIP IF NOT BACK ARROW
        JMP TTY2                DELETE CHARACTER
        SAD =377                SKIP IF NOT RUBOUT
        JMP TTY3                CLEAR 201 OUTPUT BUFFER
        JMS B.FO                SEND CHARACTER TO 201 OUTPUT BUFFER
        SKP                     DATA SET NOT CONNECTED
        JMP TTY201+2            PROCESS NEXT CHARACTER
        LAC =TTY5               GET ADDRESS OF TEXT LIST
        JMS L.T                 TYPE DIAGNOSTIC
```

```
        JMP IDLE                GET NEW COMMAND
TTY1    LAW BFETX               GET END OF TEXT CHARACTER
        JMS B.FO                SEND TO 201 OUTPUT BUFFER
        JMP *-5                 DATA SET NOT CONNECTED
        JMP IDLE                GET NEW COMMAND
TTY2    LAC =BFOB               GET ADDRESS OF OUTPUT BUFFER
        SAD BFOI                SKIP IF BUFFER NON-EMPTY
        JMP TTY201+2            IGNORE CHARACTER DELETE
        JMS B.FO                WAIT FOR ACK TO LAST RECORD
        JMP TTY1-3              DATA SET NOT CONNECTED
        LAW -2                  LOAD AC WITH -2
        TAD BFOI                COMPUTE NEW VALUE OF INPUT POINTER
        DAC BFOI                BACKSPACE OUTPUT BUFFER
        JMP TTY201+2            PROCESS NEXT CHARACTER
TTY3    JMS B.FO                WAIT FOR ACK TO LAST RECORD
        NOP                     DATA SET NOT CONNECTED
        LAC =BFOB               GET ADDRESS OF 201 OUTPUT BUFFER
        DAC BFOI                RESET INPUT POINTER
        LAC =TTY6               GET ADDRESS OF TEXT LIST
        JMS L.T                 TYPE DIAGNOSTIC
        JMP IDLE                GET NEW COMMAND
TTY4    LAW BFENQ               GET ENQUIRY
        JMP TTY1+1              SEND TO 201 OUTPUT BUFFER

TTY5    $DC !1
        $TEXT " -- DATA SET NOT CONNECTED"
TTY6    $DC 4
        $TEXT " -- DELETED"

FROM    JMS B.K                 GET KEYBOARD CHARACTER
        DZM T3                  CLEAR DATA TRANSFER POINTER
        SAD =14                 SKIP IF NOT C
        JMP FROM1               FROM CORE
        ISZ T3                  INCREMENT DATA TRANSFER POINTER
        SAD =31                 SKIP IF NOT P
        JMP FROM2               FROM PAPER TAPE
        ISZ T3                  INCREMENT DATA TRANSFER POINTER
        SAD =35                 SKIP IF NOT T
        JMP FROM3               FROM TELETYPE
        JMP IDLEQ               CANCEL COMMAND
```

```
FROM1   LAC =FROMC          GET ADDRESS OF TEXT LIST
        JMP FROM4           TYPE TEXT LIST
FROM2   LAC =FROMP          GET ADDRESS OF TEXT LIST
        SKP                 TYPE TEXT LIST
FROM3   LAC =FROMT          GET ADDRESS OF TEXT LIST
FROM4   JMS L.T             TYPE TEXT LIST
        LAC T3              GET DATA TRANSFER POINTER
        CLL+RTL             MULTIPLY BY 4
        TAD =FROM11         ADD ADDRESS OF TABLE
        DAC T3              STORE REFINED DATA TRANSFER POINTER
        LAC =FROMTO         GET ADDRESS OF TEXT LIST
        JMS L.T             TYPE TEXT LIST
        JMS B.K             GET KEYBOARD CHARACTER
        SAD =14             SKIP IF NOT C
        JMP FROM5           TO CORE
        ISZ T3              INCREMENT DATA TRANSFER POINTER
        SAD =31             SKIP IF NOT P
        JMP FROM6           TO PAPER TAPE
        ISZ T3              INCREMENT DATA TRANSFER POINTER
        SAD =35             SKIP IF NOT T
        JMP FROM7           TO TELETYPE
        ISZ T3              INCREMENT DATA TRANSFER POINTER
        SAD =15             SKIP IF NOT D
        JMP FROM8           TO DISPLAY
        JMP IDLEQ           CANCEL COMMAND
FROM5   LAC =FROMC          GET ADDRESS OF TEXT LIST
        JMP FROM9           TYPE TEXT LIST
FROM6   LAC =FROMP          GET ADDRESS OF TEXT LIST
        JMP FROM9           TYPE TEXT LIST
FROM7   LAC =FROMT          GET ADDRESS OF TEXT LIST
        SKP                 TYPE TEXT LIST
FROM8   LAC =FROMD          GET ADDRESS OF TEXT LIST
FROM9   JMS L.T             TYPE TEXT LIST
        LAC* T3             GET ADDRESS OF DATA TRANSFER
        DAC T3              SAVE TEMPORARILY
        JMP* T3             BEGIN DATA TRANSFER

FROM11  $DC TRCC
        $DC TRCP
        $DC TRCT
```

```
        $DC TRCD
        $DC TRPC
        $DC TRPP
        $DC TRPT
        $DC TRPD
        $DC TRTC
        $DC TRTP
        $DC TRTT
        $DC TRTD

FROMC   $DC 2
        $TEXT "CORE"

FROMP   $DC 4
        $TEXT "PAPER TAPE"

FROMT   $DC 3
        $TEXT "TELETYPE"

FROMD   $DC 3
        $TEXT "DISPLAY"

FROMTO  $DC 2
        $TEXT " TO "

TRCC    JMS TRBK                GET CORE BLOCK FROM KEYBOARD
        LAC =FROMTO             GET ADDRESS OF TEXT LIST
        JMS L.T                 TYPE TEXT LIST
        JMS OCTAL5              GET ADDRESS FROM KEYBOARD
        JMP IDLEQ               CANCEL COMMAND
        DAC T1                  SAVE ADDRESS
TRCC1   LAC* TRBKL              GET WORD TO BE MOVED
        DAC* T1                 STORE IN NEW LOCATION
        ISZ TRBKL               INCREMENT SOURCE POINTER
        ISZ T1                  INCREMENT SINK POINTER
        ISZ TRBKC               INCREMENT LOC COUNT & SKIP IF DONE
        JMP TRCC1               MOVE NEXT WORD
        JMP IDLE                GET NEW COMMAND

TRCP    JMS TRBK                GET CORE BLOCK FROM KEYBOARD
```

```
        LAW 100             GET ORIGIN CONTROL BIT
        DAC T3              SET CONTROL MASK
        LAC TRBKL           GET ORIGIN OF BLOCK
        JMS TRCP2           PUNCH ORIGIN
        LAW 200             GET DATA CONTROL BIT
        DAC T3              SET CONTROL MASK
TRCP1   LAC* TRBKL          GET DATA WORD
        JMS TRCP2           PUNCH DATA WORD
        ISZ TRBKL           INCREMENT POINTER
        ISZ TRBKC           INCREMENT COUNT & SKIP IF DONE
        JMP TRCP1           PUNCH NEXT WORD
        JMP IDLE            GET NEW COMMAND

TRCP2   $DC 0
        DAC T4              SAVE WORD TO BE PUNCHED
        LRS 14              SHIFT HIGH ORDER BITS INTO POSITION
        AND =77             TRUNCATE BITS FROM LINK
        XOR T3              SET CONTROL BIT
        JMS PUNCH           PUNCH IMAGE
        LAC T4              GET WORD TO BE PUNCHED
        LRS 6               SHIFT MIDDLE BITS INTO POSITION
        AND =77             TRUNCATE HIGH ORDER BITS
        XOR T3              SET CONTROL BIT
        JMS PUNCH           PUNCH IMAGE
        LAC T4              GET WORD TO BE PUNCHED
        AND =77             TRUNCATE HIGH ORDER BITS
        XOR T3              SET CONTROL BIT
        JMS PUNCH           PUNCH IMAGE
        JMP* TRCP2          RETURN

TRCT    JMS TRBK            GET CORE BLOCK FROM KEYBOARD
        LAW 17475           GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T             TYPE CARRIAGE RETURN, LINE FEED
        LAC TRBKL           GET ADDRESS TO BE TYPED
        JMS C.B6            CONVERT TO 6-BIT CODE
        TAD =770000         REMOVE HIGH ORDER ZERO
        JMS TRKT            TYPE ADDRESS
        LAW 17676           GET CODE FOR TWO SPACES
        JMS B.T             TYPE TWO SPACES
        LAW 17770           LOAD AC WITH -8
```

```
        DAC T3              SET WORD COUNTER
TRCT1   LAW 17677           GET CODE FOR ONE SPACE
        JMS B.T             TYPE SPACE
        LAC* TRBKL          GET WORD TO BE TYPED
        JMS C.B6            CONVERT TO 6-BIT CODE
        JMS TRKT            TYPE WORD
        ISZ TRBKL           INCREMENT LOCATION POINTER
        ISZ TRBKC           INCREMENT COUNT & SKIP IF DONE
        SKP                 TYPE NEXT WORD
        JMP IDLE            GET NEW COMMAND
        ISZ T3              SKIP IF END OF LINE
        JMP TRCT1           TYPE NEXT WORD
        JMP TRCT+1          BEGIN NEW LINE

TRCD    JMS TRBK            GET CORE BLOCK FROM KEYBOARD
        LAC TRBKC           GET WORD COUNT
        TAD =100            MAKE POSITIVE IF NOT TOO LARGE
        SMA                 SKIP IF TOO LARGE
        JMP *+3             WORD COUNT OK
        LAW 17700           LOAD AC WITH -64
        DAC TRBKC           ADJUST WORD COUNT
        JMS TRD1            INITIALIZE TEXT LIST FOR DISPLAY
TRCD1   LAC TRBKL           GET ADDRESS TO BE DISPLAYED
        JMS C.B6            CONVERT TO 6-BIT CODE
        TAD =770000         REMOVE HIGH ORDER ZERO
        JMS TRD2            PUT HIGH ORDER DIGITS IN TEXT LIST
        LACQ                GET LOW ORDER DIGITS
        JMS TRD2            PUT LOW ORDER DIGITS IN TEXT LIST
        LAW 17676           GET CODE FOR TWO SPACES
        JMS TRD2            PUT IN TEXT LIST
        LAW 17770           LOAD AC WITH -8
        DAC T3              SET WORD COUNTER
TRCD2   LAW 17677           GET CODE FOR ONE SPACE
        JMS TRD2            PUT IN TEXT LIST
        LAC* TRBKL          GET WORD TO BE DISPLAYED
        JMS C.B6            CONVERT TO 6-BIT CODE
        JMS TRD2            PUT HIGH ORDER DIGITS IN TEXT LIST
        LACQ                GET LOW ORDER DIGITS
        JMS TRD2            PUT LOW ORDER DIGITS IN TEXT LIST
        ISZ TRBKL           INCREMENT LOCATION POINTER
```

```
        ISZ  TRBKC          INCREMENT COUNT & SKIP IF DONE
        JMP  TRCD3          PREPARE NEXT WORD
        CLC                 GET THREE NULL CHARACTERS
        DAC  T3             NULLIFY ACCUMULATED CHARACTERS
        JMS  TRD3           DISPLAY TEXT LIST
        JMP  IDLE           GET NEW COMMAND
TRCD3   ISZ  T3             SKIP IF END OF LINE
        JMP  TRCD2          PREPARE NEXT WORD
        LAC  =747575        GET CARRIAGE RETURN, LINE FEED CODE
        JMS  TRD2           PUT IN TEXT LIST
        JMP  TRCD1          BEGIN NEW LINE

TRPC    JMS  READ           READ ONE TAPE IMAGE
        SNA                 SKIP IF NOT END OF RECORD
        JMP  IDLE           GET NEW COMMAND
        DAC  T3             SAVE TAPE LINE
        AND  =300           GET CONTROL BITS
        SAD  =100           SKIP IF NOT ORIGIN
        JMP  TRPC1          COMPLETE ORIGIN
        SAD  =200           SKIP IF NOT BINARY DATA
        JMP  TRPC2          COMPLETE DATA WORD
        JMS  READ           READ A TAPE IMAGE
        SNA                 SKIP IF NOT END OF RECORD
        JMP  TRPC           RESTART DATA TRANSFER
        JMP  *-3            IGNORE TAPE IMAGE
TRPC1   JMS  TRPC3          FINISH READING ORIGIN
        DAC  T4             SET ORIGIN
        JMP  TRPC           GET NEXT WORD FROM TAPE
TRPC2   JMS  TRPC3          FINISH READING DATA WORD
        DAC* T4             LOAD DATA WORD
        ISZ  T4             INCREMENT LOCATION COUNTER
        JMP  TRPC           GET NEXT WORD FROM TAPE

TRPC3   $DC  0
        JMS  READ           GET SECOND IMAGE FROM TAPE
        LRS  6              SHIFT DATA BITS INTO MQ
        LAC  T3             GET HIGH ORDER 6 BITS
        LLS  6              SHIFT HIGH ORDER 12 BITS INTO AC
        DAC  T3             SAVE HIGH ORDER 12 BITS
        JMS  READ           GET THIRD IMAGE FROM TAPE
```

```
        LRS 6               SHIFT DATA BITS INTO MQ
        LAC T3              GET HIGH ORDER 12 BITS
        LLS 6               SHIFT COMPLETED WORD INTO AC
        JMP* TRPC3          RETURN

TRPP    JMS READ            GET IMAGE FROM PAPER TAPE
        SAD =377            SKIP IF NOT END-OF-TAPE GARBAGE
        JMP TRPP            RESTART DATA TRANSFER
        DAC T3              SAVE TEMPORARILY
        AND =300            GET CONTROL BITS
        SAD =300            SKIP IF NOT ALPHANUMERIC
        JMS TRPP3           PUNCH END-OF-RECORD MARK
        LAC T3              GET IMAGE READ
TRPP1   JMS PUNCH           PUNCH IMAGE
        JMS READ            GET IMAGE FROM PAPER TAPE
        SNA                 SKIP IF NOT END OF RECORD
        JMP TRPP2           PUNCH END-OF-RECORD IF NECESSARY
        DAC T3              SAVE TEMPORARILY
        JMP TRPP1           PUNCH IMAGE
TRPP2   LAC T3              GET LAST IMAGE PUNCHED
        AND =300            GET CONTROL BITS
        SAD =300            SKIP IF NOT ALPHANUMERIC
        JMS TRPP3           PUNCH END-OF-RECORD MARK
        JMP IDLE            GET NEW COMMAND
TRPP3   $DC 0
        CLA                 GET END-OF-RECORD MARK
        JMS PUNCH           PUNCH END-OF-RECORD MARK
        JMP* TRPP3          RETURN
```

```
        $TITLE          IDLE-TIME TASK (CONTINUED)

TRPT    JMS READ        GET IMAGE FROM PAPER TAPE
        SAD =377        SKIP IF NOT END-OF-TAPE GARBAGE
        JMP TRPT        RESTART DATA TRANSFER
        DAC T3          SAVE TEMPORARILY
        AND =300        GET CONTROL BITS
        SAD =300        SKIP IF BINARY INFORMATION
        JMP TRPT1       RECORD IS ALPHANUMERIC
        JMS READ        GET IMAGE FROM PAPER TAPE
        SNA             SKIP IF NOT END OF RECORD
        JMP TRPT        TRY TRANSFER AGAIN
        JMP *-3         GET NEXT IMAGE
TRPT1   LAW 17475       GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T         TYPE CARRIAGE RETURN, LINE FEED
        LAC T3          GET FIRST IMAGE FROM TAPE
TRPT2   XOR =777400     PRECEDE WITH NULL CHARACTERS
        JMS B.T         TYPE CHARACTER FROM TAPE
        JMS READ        GET IMAGE FROM TAPE
        SNA             SKIP IF NOT END OF RECORD
        JMP IDLE        GET NEW COMMAND
        JMP TRPT2       TYPE CHARACTER

TRPD    JMS READ        READ IMAGE FROM TAPE
        SNA             SKIP IF NOT END-OF-RECORD CHARACTER
        JMP TRPD        RESTART DATA TRANSFER
        SAD =375        SKIP IF NOT LINE FEED
        JMP TRPD        RESTART DATA TRANSFER
        SAD =377        SKIP IF NOT END-OF-TAPE GARBAGE
        JMP TRPD        RESTART DATA TRANSFER
        DAC T6          SAVE TEMPORARILY
        AND =300        GET CONTROL BITS
        SAD =300        SKIP IF BINARY
        JMP TRPD1       RECORD OK
        JMS READ        READ IMAGE FROM TAPE
        SNA             SKIP IF NOT END OF RECORD
        JMP TRPD        TRY TRANSFER AGAIN
        JMP *-3         IGNORE IMAGE
TRPD1   JMS TRD1        INITIALIZE TEXT LIST
        LAW 17766       LOAD AC WITH -10
```

```
        DAC T4              SET LINE COUNTER
        LAW 17676           LOAD AC WITH -66
        DAC T5              SET CHARACTER COUNTER
        LAC T6              GET FIRST CHARACTER
        JMP TRPD2+3         ADD TO TEXT LIST
TRPD2   LAW 17676           LOAD AC WITH -66
        DAC T5              SET CHARACTER COUNTER
        JMS READ            READ IMAGE FROM TAPE
        SAD =374            SKIP IF NOT CARRIAGE RETURN
        JMP TRPD3           TERMINATE LINE
        SNA                 SKIP IF NOT END OF RECORD
        JMP TRPD4           TERMINATE TRANSFER
        JMS TRD4            ADD CHARACTER TO TEXT LIST
        ISZ T5              INCREMENT CHAR COUNT & SKIP IF DONE
        JMP TRPD2+2         GET NEXT CHARACTER
TRPD3   ISZ T4              INCREMENT COUNTER & SKIP IF DONE
        SKP                 GET ANOTHER LINE
        JMP TRPD4           TERMINATE TRANSFER
        LAW 74              GET CARRIAGE RETURN
        JMS TRD4            ADD TO TEXT LIST
        LAW 75              GET LINE FEED
        JMS TRD4            ADD TO TEXT LIST
        JMP TRPD2           BEGIN NEW LINE
TRPD4   JMS TRD3            DISPLAY TEXT LIST
        JMP IDLE            GET NEW COMMAND

TRTC    LAW 17475           GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T             TYPE IT
        JMS OCTAL5          GET ADDRESS FROM KEYBOARD
        JMP TRTC4           INTERPRET AS COMMAND
        DAC T5              STORE ADDRESS
TRTC1   LAW 17677           GET CODE FOR ONE SPACE
        JMS B.T             TYPE IT
        LAC* T5             GET CURRENT CONTENT OF WORD
        JMS C.B6            CONVERT TO 6-BIT CODE
        JMS TRKT            TYPE CURRENT CONTENTS
        LAW 17677           GET CODE FOR ONE SPACE
        JMS B.T             TYPE IT
        JMS OCTAL6          GET NEW CONTENTS FROM KEYBOARD
        JMP TRTC3           DETERMINE NATURE OF FAILURE
```

```
        DAC* T5         STORE NEW CONTENTS
TRTC2   ISZ T5          INCREMENT STORED ADDRESS
        LAW 17475       GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T         TYPE CARRIAGE RETURN, LINE FEED
        LAC T5          GET CURRENT ADDRESS
        JMS C.B6        CONVERT TO 6-BIT CODE
        TAD =770000     REMOVE HIGH ORDER ZERO
        JMS TRKT        TYPE CURRENT ADDRESS
        JMP TRTC1       TYPE CONTENTS OF CURRENT LOCATION
TRTC3   SAD =74         SKIP IF NOT CARRIAGE RETURN
        JMP TRTC2       LEAVE WORD UNCHANGED
        JMP TRTC        BEGIN INTERPRETATION OF NEW BLOCK
TRTC4   DAC T3          SAVE KEYBOARD CHARACTER
        LAW 10          GET TELEPRINTER MASK
        JMS T.R         RELEASE TELEPRINTER
        JMP IDLE+6      INTERPRET CHARACTER AS COMMAND

TRTP    CLA             GET END-OF-RECORD MARK
        JMS PUNCH       PUNCH IT
        LAW 17475       GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T         TYPE CARRIAGE RETURN, LINE FEED
TRTP1   JMS ECHO        ECHO KEYBOARD CHARACTER
        SAD =77         SKIP IT NOT NULL CHARACTER
        JMP TRTP2       TERMINATE TRANSFER
        XOR =300        SET ALPHANUMERIC CONTROL BITS
        JMS PUNCH       PUNCH CHARACTER
        JMP TRTP1       GET NEXT CHARACTER
TRTP2   CLA             GET END-OF-RECORD MARK
        JMS PUNCH       PUNCH IT
        JMP IDLE        GET NEW COMMAND

TRTT    LAW 17475       GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T         TYPE IT
        JMS ECHO        ECHO KEYBOARD CHARACTER
        SAD =77         SKIP IF NOT NULL CHARACTER
        JMP IDLE        GET NEW COMMAND
        JMP *-3         GET NEXT CHARACTER

TRTD    LAW 17766       LOAD AC WITH -10
        DAC T4          INITIALIZE LINE COUNTER
```

```
        JMS TRD1            INITIALIZE TEXT LIST
TRTD1   LAW 17475           GET CARRIAGE RETURN, LINE FEED CODE
        JMS B.T             TYPE CARRIAGE RETURN, LINE FEED
        LAW 17700           LOAD AC WITH -64
        DAC T5              INITIALIZE CHARACTER COUNTER
TRTD2   JMS ECHO            ECHO KEYBOARD CHARACTER
        SAD =77             SKIP IF NOT NULL CHARACTER
        JMP TRTD4           DISPLAY TEXT LIST
        SAD =74             SKIP IF NOT CARRIAGE RETURN
        JMP TRTD3           TERMINATE LINE
        JMS TRD4            ADD CHARACTER TO TEXT LIST
        ISZ T5              SKIP IF END OF LINE
        JMP TRTD2           GET NEXT CHARACTER
TRTD3   ISZ T4              INCREMENT LINE COUNT & SKIP IF DONE
        SKP                 TERMINATE LINE
        JMP TRTD4           TERMINATE TRANSFER
        LAW 74              GET CARRIAGE RETURN CODE
        JMS TRD4            ADD TO TEXT LIST
        LAW 75              GET LINE FEED CODE
        JMS TRD4            ADD TO TEXT LIST
        LAW 75              GET LINE FEED CODE
        JMS TRD4            ADD TO TEXT LIST
        JMP TRTD1           BEGIN NEW LINE
TRTD4   JMS TRD3            DISPLAY TEXT LIST
        JMP IDLE            GET NEW COMMAND

ECHO    $DC 0
        JMS B.K             GET CHARACTER FROM KEYBOARD
        DAC T6              SAVE TEMPORARILY
        XOR =777700         PRECEDE WITH NULL CHARACTERS
        JMS B.T             ECHO CHARACTER ON TELEPRINTER
        LAC T6              GET CHARACTER FOR RETURN
        JMP* ECHO           RETURN

PUNCH   $DC 0
        JMS B.P             SEND IMAGE TO PUNCH
        SKP                 PUNCH OUT OF TAPE
        JMP* PUNCH          RETURN
        LAC =PUNCH1         GET ADDRESS OF TEXT LIST
        JMS L.T             TYPE TEXT LIST
```

```
        JMP IDLE                GET NEW COMMAND

PUNCH1  $DC 7
        $DC 747531
        $TEXT "UNCH OUT OF TAPE"

READ    $DC 0
        JMS B.R                 GET IMAGE FROM READER BUFFER
        SKP                     READER OUT OF TAPE
        JMP* READ               RETURN
        LAC =READ1              GET ADDRESS OF TEXT LIST
        JMS L.T                 TYPE TEXT LIST
        JMP IDLE                GET NEW COMMAND

READ1   $DC 7
        $DC 747533
        $TEXT "EADER OUT OF TAPE"

TRD1    $DC 0
        LAC 27                  GET POINTER TO LEAF
        LMQ                     SET UP PARAMETER
        LAC 26                  GET POINTER TO LEVEL
        SZA                     SKIP IF NO LEVEL
        JMS S.TR                REMOVE LEAF FROM LEVEL
        NOP                     LEAF OR LEVEL DIDN'T EXIST
        LAC 27                  GET ADDRESS OF LEAF
        SZA                     SKIP IF NO LEAF
        JMS L.L                 DESTROY LEAF
        DZM 27                  INDICATE NO LEAF
        DZM TRDT                CLEAR TEXT LIST COUNT
        LAC =TRDT               GET ADDRESS OF TEXT LIST
        DAC TRDP                INITIALIZE TEXT LIST POINTER
        CLC                     GET 3 NULL CHARACTERS
        DAC T3                  STORE NULL CHARACTERS
        JMP* TRD1               RETURN

TRD2    $DC 0
        ISZ TRDT                INCREMENT TEXT LIST COUNT
        ISZ TRDP                INCREMENT TEXT LIST POINTER
        DAC* TRDP               STORE NEW TEXT WORD
```

```
        JMP* TRD2               RETURN

TRD3    $DC 0
        LAC T3                  GET REMAINING CHARACTERS
        JMS TRD2                PUT IN TEXT LIST
        LAC 26                  GET ADDRESS OF LEVEL
        SNA                     SKIP IF LEVEL EXISTS
        JMP TRD31               DISPLAY STORAGE EXCEEDED
        LAC =TRDT               GET ADDRESS OF TEXT LIST
        JMS L.D                 CREATE TEXT LEAF
        JMP TRD31               STORAGE EXCEEDED
        DAC 27                  SAVE ADDRESS OF LEAF
        LMQ                     SET UP PARAMETER
        LAC 26                  GET ADDRESS OF LEVEL
        JMS S.TI                INSERT LEAF
        JMP TRD31               STORAGE EXCEEDED
        JMP* TRD3               RETURN
TRD31   LAC =*+3                GET ADDRESS OF TEXT LIST
        JMS L.T                 TYPE DIAGNOSTIC
        JMP IDLE                GET NEW COMMAND
        $DC 12
        $DC 747577
        $TEXT "NOT ENOUGH DISPLAY STORAGE"

TRD4    $DC 0
        LRS 6                   SHIFT CHARACTER INTO MQ
        LAC T3                  GET PREVIOUS CHARACTERS
        LLS 6                   SHIFT ALL CHARACTERS INTO AC
        DAC T3                  SAVE CHARACTERS
        AND =770000             GET HIGH ORDER CHARACTER
        SAD =770000             SKIP IF NOT NULL
        JMP* TRD4               RETURN
        LAC T3                  GET WORD OF 3 CHARACTERS
        JMS TRD2                ADD TO TEXT LIST
        CLC                     GET 3 NULL CHARACTERS
        DAC T3                  STORE NULL CHARACTERS
        JMP* TRD4               RETURN

TRDT    $DS 351
```

```
STC     $DC 0
        LAC =D                  GET ADDRESS OF HIGHEST ACTIVE LEVEL
        DAC DHAL+7              REMOVE ALL NODES FROM HAL
        JMS DW                  WAIT FOR DISPLAY TO RECOVER
        LAW STORE               GET INITIAL COUNTER VALUE
        DAC T1                  SET POINTER & COUNTER
        IOT 7704                LEAVE EXTEND MODE
        DZM* T1                 CLEAR STORAGE LOCATION
        ISZ T1                  INCREMENT POINTER & COUNTER
        JMP *-2                 CLEAR NEXT STORAGE LOCATION
        IOT 7702                ENTER EXTEND MODE
        JMP* STC                RETURN

OCTAL1  $DC 0
        JMS B.K                 GET KEYBOARD CHARACTER
        TAD =-10                MAKE NEGATIVE IF OCTAL
        SPA                     SKIP IF NOT OCTAL DIGIT
        JMP *+3                 OCTAL DIGIT TYPED
        TAD =10                 RESTORE CHARACTER
        JMP* OCTAL1             INDICATE FAILURE
        DAC T3                  SAVE OCTAL INFORMATION
        XOR =70                 CONVERT TO 6-BIT CODE
        JMS B.T                 TYPE OCTAL DIGIT
        LAC T3                  GET OCTAL INFORMATION
        LRS 3                   SHIFT DIGIT INTO MQ
        LAC T4                  GET RECORDED DIGITS
        LLS 3                   CONCATENATE NEW DIGIT
        DAC T4                  RECORD NEW WORD
        ISZ OCTAL1              INDICATE SUCCESS
        JMP* OCTAL1             RETURN

OCTAL5  $DC 0
        DZM T4                  CLEAR OCTAL RECORDING WORD
        JMS OCTAL1              GET OCTAL DIGIT FROM KEYBOARD
        JMP* OCTAL5             NON-OCTAL CHARACTER TYPED
        JMS OCTAL1              GET OCTAL DIGIT FROM KEYBOARD
        JMP* OCTAL5             NON-OCTAL CHARACTER TYPED
        JMS OCTAL1              GET OCTAL DIGIT FROM KEYBOARD
        JMP* OCTAL5             NON-OCTAL CHARACTER TYPED
        JMS OCTAL1              GET OCTAL DIGIT FROM KEYBOARD
```

```
        JMP* OCTAL5         NON-OCTAL CHARACTER TYPED
        JMS OCTAL1          GET OCTAL CHARACTER FROM KEYBOARD
        JMP* OCTAL5         NON-OCTAL CHARACTER TYPED
        ISZ OCTAL5          INDICATE SUCCESS
        JMP* OCTAL5         RETURN

OCTAL6  $DC 0
        JMS OCTAL5          GET 5 OCTAL DIGITS FROM KEYBOARD
        JMP* OCTAL6         NON-OCTAL CHARACTER TYPED
        JMS OCTAL1          GET OCTAL DIGIT FROM KEYBOARD
        JMP* OCTAL6         NON-OCTAL CHARACTER TYPED
        ISZ OCTAL6          INDICATE SUCCESS
        JMP* OCTAL6         RETURN

TRKT    $DC 0
        DAC T1              SAVE HIGH ORDER DIGITS
        LACQ                GET LOW ORDER DIGITS
        DAC T6              SAVE LOW ORDER DIGITS
        LAC T1              GET HIGH ORDER DIGITS
        JMS B.T             TYPE HIGH ORDER DIGITS
        LAC T6              GET LOW ORDER DIGITS
        JMS B.T             TYPE LOW ORDER DIGITS
        JMP* TRKT           RETURN

TRBK    $DC 0
        LAC =TRBKF          GET ADDRESS OF TEXT LIST
        JMS L.T             TYPE TEXT LIST
        JMS OCTAL5          GET LOW ADDRESS FROM KEYBOARD
        JMP IDLEQ           CANCEL COMMAND
        DAC TRBKL           STORE LOW ADDRESS
        LAW 16277           GET COMMA CODE
        JMS B.T             TYPE COMMA
        JMS OCTAL5          GET HIGH ADDRESS FROM KEYBOARD
        JMP IDLEQ           CANCEL COMMAND
        CMA                 FORM ONE'S COMPLEMENT
        TAD TRBKL           ADD LOW ADDRESS
        SMA                 SKIP IF PROPERLY ORDERED ADDRESSES
        JMP IDLEQ           CANCEL COMMAND
        DAC TRBKC           STORE LOCATION COUNT
        LAW 15177           GET RIGHT PARENTHESIS CODE
```

```
        JMS B.T                 TYPE RIGHT PARENTHESIS
        JMP* TRBK               RETURN

TRBKF   $DC 3
        $DC 747513
        $TEXT "LOCK("
```

```
          STITLE                    HIGHEST ACTIVE LEVEL

D         $DC 757
          $DC 6201
          $DC 6301
          $DC 1400
DHAL      $DC DWT
          $DC 2010
          $DC XP
          $DC 1105
          $DC 1000
          $DC 5000
          $DC 2001
          $DC D

          $END
```

APPENDIX B -- SUMMARY OF SYSTEM SUBROUTINES

THE FOLLOWING TABLE OF SYSTEM SUBROUTINES IS PROVIDED AS A REFERENCE TO FACILITATE THE WRITING OF USER PROGRAMS. THE VARIOUS COLUMNS ARE INTERPRETED AS FOLLOWS:

NAME -- SYMBOLIC NAME OF THE SYSTEM SUBROUTINE

ENTRY POINT -- ADDRESS AT WHICH THE SUBROUTINE STARTS

SECTION -- SECTION OF THE REPORT IN WHICH THE SUBROUTINE IS DESCRIBED

FAILURE RETURN -- WHETHER OR NOT A FAILURE RETURN EXISTS

DELAY POSSIBLE -- WHETHER OR NOT OTHER TASKS MAY BE EXECUTED BEFORE THE SUBROUTINE RETURNS

NAME	ENTRY POINT	SECTION	FAILURE RETURN	DELAY POSSIBLE
B.FI	140	3.4.1	YES	YES
B.FO	142	3.4.1	YES	YES
B.K	150	3.4.3	NO	YES
B.P	146	3.4.2	YES	YES
B.R	144	3.4.2	YES	YES
B.T	152	3.4.3	NO	YES
C.6A	130	3.3	NO	NO
C.A6	132	3.3	NO	NO
C.B6	126	3.3	NO	NO
C.BC	136	3.3	NO	NO
C.CB	134	3.3	NO	NO
D.A	206	3.7	NO	NO
D.D	202	3.7	NO	NO
D.E	200	3.7	NO	NO
D.O	214	3.7	YES	NO
D.P	204	3.7	NO	NO
D.X	212	3.7	NO	NO
D.Y	210	3.7	NO	NO

NAME	ENTRY POINT	SECTION	FAILURE RETURN	DELAY POSSIBLE
L.D	272	3.11	YES	NO
L.L	274	3.11	NO	NO
L.T	270	3.11	NO	YES
N.A	154	3.5	NO	YES
N.C	156	3.5	NO	YES
N.D1	160	3.5	NO	NO
N.D2	162	3.5	NO	NO
N.D3	164	3.5	NO	NO
P.D	172	3.6	NO	NO
P.E	170	3.6	NO	NO
P.R	174	3.6	NO	NO
P.S	176	3.6	NO	NO
P.T	166	3.6	NO	NO
Q.A	102	3.1	YES	NO
Q.C	100	3.1	NO	NO
Q.F	106	3.1	YES	NO
Q.I	104	3.1	YES	NO
S.LBD	254	3.10	NO	NO
S.LBE	252	3.10	NO	NO
S.LC	256	3.10	NO	NO
S.LH	242	3.10	NO	NO
S.LL	264	3.10	NO	NO
S.LN	266	3.10	NO	NO
S.LP	250	3.10	NO	NO
S.LS	262	3.10	NO	NO
S.LU	260	3.10	NO	NO
S.LX	246	3.10	NO	YES
S.LY	244	3.10	NO	YES
S.TD	234	3.9	YES	NO
S.TI	236	3.9	YES	NO
S.TL	232	3.9	YES	NO
S.TR	240	3.9	YES	YES
T.A	116	3.2	NO	YES
T.F	114	3.2	--	--
T.L	122	3.2	--	--
T.P	112	3.2	--	--
T.R	120	3.2	NO	NO

NAME	ENTRY POINT	SECTION	FAILURE RETURN	DELAY POSSIBLE
T.S	110	3.2	NO	NO
T.U	124	3.2	--	--
X.I	216	3.8	NO	NO
X.R	220	3.8	NO	NO
X.S	224	3.8	NO	NO
X.T	222	3.8	NO	NO
X.X	230	3.8	NO	NO
X.Y	226	3.8	NO	NO

APPENDIX C -- SUMMARY OF IOT INSTRUCTIONS

STATUS WORDS

ALL BITS WHOSE INTERPRETATIONS ARE NOT SPECIFIED BELOW ARE NOT USED.

PDP-9 I/O STATUS

BIT	INTERPRETATION
0	INTERRUPTS ARE ENABLED
1	READER FLAG
2	PUNCH FLAG
3	KEYBOARD FLAG
4	TELEPRINTER FLAG
6	CLOCK FLAG
7	CLOCK ENABLED
8	READER OUT-OF-TAPE FLAG
9	PUNCH OUT-OF-TAPE FLAG
11	201 DATAPHONE TRANSMIT FLAG
12	201 DATAPHONE RECEIVE FLAG

201 DATAPHONE STATUS

BIT	INTERPRETATION
0	INTERRUPT PENDING
1	DATA LOST
2	PARITY ERROR
3	REQUEST TO SEND
4	TRANSMIT REQUEST
5	CLEAR TO SEND
6	CHECK PARITY
7	TEXT MODE
8	SET READY
9	TERMINAL READY
10	RING
11	CARRIER DETECTED
12	FRAME SIZE REGISTER BIT 0
13	FRAME SIZE REGISTER BIT 1
14	FRAME SIZE REGISTER BIT 2
15	FRAME SIZE REGISTER BIT 3

BIT	INTERPRETATION
16	TRANSMIT STATE
17	RECEIVE STATE

DISPLAY STATUS WORD 1

BIT	INTERPRETATION
6	LIGHT PEN FLAG
7	VERTICAL EDGE FLAG
8	HORIZONTAL EDGE FLAG
9	INTERNAL STOP FLAG
10	SECTOR 0 FLAG (DISPLAY COORDINATES ARE ON SCREEN)
11	CONTROL STATE
12	MANUAL INTERRUPT FLAG
13	PUSH BUTTON FLAG
14	DISPLAY INTERRUPT PENDING
15	BREAK FIELD REGISTDR BIT 0
16	BREAK FIELD REGISTER BIT 1
17	BREAK FIELD REGISTER BIT 2

DISPLAY STATUS WORD 2

BIT	INTERPRETATION
6	0 -- LEFT HAND INCREMENT BEING EXECUTED 1 -- RIGHT HAND INCREMENT BEING EXECUTED
7	LIGHT PEN ENABLED
8	BIT 0 OF Y POSITION REGISTER
9	BIT 0 OF X POSITION REGISTER
10	SCALE BIT 0
11	SCALE BIT 1
12	MODE BIT 0
13	MODE BIT 1
14	MODE BIT 2
15	INTENSITY BIT 0
16	INTENSITY BIT 1
17	INTENSITY BIT 2

DISPLAY INITIAL CONDITIONS

BIT	INTERPRETATION
6	ENABLE EDGE FLAG INTERRUPT
7	ENABLE LIGHT PEN FLAG INTERRUPT
8	0 -- DO NOT DISABLE LIGHT PEN AFTER RESUMING DISPLAY 1 -- ENABLE LIGHT PEN ACCORDING TO BIT 9
9	0 -- ENABLE LIGHT PEN AFTER FIRST DATA REQUEST AFTER RESUMING DISPLAY 1 -- DO NOT ENABLE LIGHT PEN AFTER RESUMING DISPLAY
10	BIT 0 OF Y DIMENSION
11	BIT 1 OF Y DIMENSION
12	BIT 0 OF X DIMENSION
13	BIT 1 OF X DIMENSION
14	INTENSIFY ALL POINTS
15	INHIBIT EDGE FLAGS
16	ENABLE PUSH BUTTON INTERRUPT
17	ENABLE INTERNAL STOP INTERRUPT

BREAK FIELD LOAD PARAMETER

BIT	INTERPRETATION
6	LOAD BREAK FIELD ACCORDING TO BITS 7-9
7	BREAK FIELD BIT 0
8	BREAK FIELD BIT 1
9	BREAK FIELD BIT 2
10	LOAD PUSH BUTTONS ACCORDING TO BITS 11-17
11	0 -- LOAD PUSH BUTTONS 0-5 1 -- LOAD PUSH BUTTONS 6-11
12	PUSH BUTTON 0 OR 6
13	PUSH BUTTON 1 OR 7
14	PUSH BUTTON 2 OR 8
15	PUSH BUTTON 3 OR 9
16	PUSH BUTTON 4 OR 10
17	PUSH BUTTON 5 OR 11

IOT INSTRUCTIONS

EACH IOT INSTRUCTION IS FORMED BY ADDING THE CODE FROM THE TABLE BELOW TO 700000. THE AC MAY BE CLEARED AT EVENT TIME 1 OF THE IOT INSTRUCTION BY SETTING BIT 14 IN THE INSTRUCTION.

CODE	FUNCTION
0002	ENABLE INTERRUPTS
0042	DISABLE INTERRUPTS
0001	SKIP IF CLOCK FLAG IS SET
0004	CLEAR CLOCK FLAG AND DISABLE CLOCK
0044	CLEAR CLOCK FLAG AND ENABLE CLOCK
0101	SKIP IF READER FLAG IS SET
0102	CLEAR READER FLAG, INCLUSIVE OR CONTENT OF READER BUFFER INTO AC
0104	SELECT READER IN ALPHANUMERIC MODE
0144	SELECT READER IN BINARY MODE
0201	SKIP IF PUNCH FLAG IS SET
0202	CLEAR PUNCH FLAG
0206	PUNCH TAPE IMAGE FROM BITS 10-17 OF AC
0244	PUNCH TAPE IMAGE IN BINARY MODE FROM BITS 12-17 OF AC
0301	SKIP IF KEYBOARD FLAG IS SET
0302	OR CONTENT OF KEYBOARD BUFFER INTO BITS 10-17 OF AC
0304	OR I/O STATUS WORD INTO AC

CODE	FUNCTION
0401	SKIP IF TELEPRINTER FLAG IS SET
0402	CLEAR TELEPRINTER FLAG
0406	LOAD TELEPRINTER BUFFER FROM BITS 10-17 OF THE AC
0501	OR DISPLAY PUSH-DOWN POINTER INTO BITS 6-17 OF THE AC
0502	OR BITS 1-12 OF THE DISPLAY CONTROL X POSITION REGISTER INTO BITS 6-17 OF THE AC
0601	OR BITS 3-14 OF THE DISPLAY ADDRESS COUNTER INTO BITS 6-17 OF THE AC
0602	OR DISPLAY STATUS WORD 1 INTO BITS 6-17 OF THE AC
0621	OR PUSH BUTTONS 0-11 INTO BITS 6-17 OF THE AC
0642	SKIP IF THE LIGHT PEN FLAG IS SET
0645	SET DISPLAY PUSH DOWN POINTER FROM BITS 6-17 OF THE AC
0665	SET DISPLAY INITIAL CONDITIONS FROM BITS 6-17 OF THE AC
0701	SKIP IF DISPLAY EXTERNAL STOP FLAG IS SET
0702	SKIP IF EITHER THE VERTICAL OR HORIZONTAL EDGE FLAG IS SET
0704	STOP DISPLAY (EXTERNAL)
0705	LOAD BREAK FIELD AND/OR PUSH BUTTONS FROM THE BREAK FIELD PARAMETER IN BITS 6-17 OF THE AC
0721	SKIP IF DISPLAY INTERNAL STOP FLAG IS SET
0722	SKIP IF MANUAL INTERRUPT FLAG IS SET

CODE	FUNCTION
1103	SET THE A/D CONVERTER MULTIPLEXOR TO THE CHANNEL SPECIFIED IN BITS 12-17 OF THE AC
1201	INCREMENT THE A/D CONVERTER MULTIPLEXOR CHANNEL NUMBER (CHANNEL 0 FOLLOWS CHANNEL 77)
1202	OR A/D CONVERTER MULTIPLEXOR CHANNEL NUMBER INTO BITS 12-17 OF THE AC
1301	SKIP IF THE A/D CONVERTER FLAG IS SET
1302	OR A/D CONVERTER BUFFER INTO BITS 0-11 OF THE AC
1304	SELECT THE A/D CONVERTER
1401	SKIP IF THE DATAPHONE TRANSMIT FLAG IS SET
1402	OR THE DATAPHONE STATUS WORD INTO THE AC
1404	INVERT THE DATAPHONE STATUS BITS WHEREVER A 1 APPEARS IN THE CORRESPONDING POSITION IN THE AC
1421	SKIP IF DATAPHONE MASK SKIP FLAG IS SET
1422	SET THE DATAPHONE MASK SKIP FLAG IF ALL BITS IN THE DATAPHONE STATUS WORD ARE 1'S WHEREVER A 1 APPEARS IN THE CORRESPONDING POSITION IN THE AC
1424	CLEAR DATAPHONE MASK SKIP FLAG
1441	SKIP IF THE DATAPHONE RECEIVE FLAG IS SET
1442	CLEAR THE DATAPHONE TRANSMIT AND RECEIVE FLAGS
1444	CLEAR ALL DATAPHONE FLAGS AND REGISTERS
1601	CLEAR DISPLAY FLAGS

CODE	FUNCTION
1602	OR BITS 1-12 OF THE DISPLAY Y POSITION REGISTER INTO BITS 6-17 OF THE AC
1604	RESUME DISPLAY AFTER INTERNAL STOP
1605	INITIALIZE DISPLAY AT ADDRESS GIVEN IN BITS 6-17 OF THE AC
1622	OR DISPLAY STATUS WORD 2 INTO BITS 6-17 OF THE AC
3301	SKIP IF THE TELETYPE IS CONNECTED
3302	CLEAR ALL FLAGS
3344	RESTORE THE LINK AND EXTEND MODE STATUS FROM INFORMATION CONTAINED IN THE LOCATION WHOSE ADDRESS IS GIVEN IN BITS 5-17 OF THE FOLLOWING WORD IN MEMORY
5101	LOAD D/A CONVERTER CHANNEL #1 FROM BITS 0-11 OF THE AC
5102	LOAD D/A CONVERTER CHANNEL #2 FROM BITS 0-11 OF THE AC
5104	LOAD D/A CONVERTER CHANNEL #3 FROM BITS 0-11 OF THE AC
7701	SKIP IF IN EXTEND MODE
7702	ENTER EXTEND MODE
7704	LEAVE EXTEND MODE

APPENDIX D -- ASSEMBLY LANGUAGE

THE ASSEMBLY LANGUAGE WHICH IS USED IN THE EXAMPLES IN THE REPORT IS THE SOURCE LANGUAGE FOR THE ASSEMBLER (TO BE DESCRIBED IN A FORTHCOMING REPORT) WHICH RUNS UNDER THE EXECUTIVE SYSTEM. THIS LANGUAGE IS DESCRIBED BRIEFLY BELOW.

ALL MNEMONICS ARE FROM ONE TO SIX CHARACTERS LONG. THE FIRST CHARACTER MUST BE AN ALPHABETIC CHARACTER OR A PERIOD (.), AND ALL OTHER CHARACTERS MUST BE ALPHANUMERIC OR PERIODS. A MNEMONIC MAY REPRESENT ANY ONE OF THE FOLLOWING ENTITIES:

(1) A PROGRAM SYMBOL (I. E., A SYMBOL WHOSE VALUE IS USED TO COMPUTE THE OPERAND OF AN INSTRUCTION),

(2) AN INSTRUCTION CODE, OR

(3) A PSEUDO-OP (I. E., AN INSTRUCTION TO THE ASSEMBLER).

IF A MNEMONIC IS USED TO REPRESENT MORE THAN ONE OF THESE ENTITIES, THE ASSEMBLER WILL RESOLVE THE AMBIGUITY FROM CONTEXT.

ALL NUMBERS ARE INTERPRETED AS OCTAL NUMBERS. NUMBERS MAY REPRESENT VALUES OF PROGRAM SYMBOLS ONLY.

A SOURCE LINE IS COMPOSED OF UP TO FOUR FIELDS. EACH FIELD IS DELIMITED BY SPACES. (SEVERAL CONSECUTIVE SPACES ARE INTERPRETED AS A SINGLE SPACE BY THE ASSEMBLER, EXCEPT IN TEXT PSEUDO-OP OPERANDS.) THE FOUR POSSIBLE FIELDS (FROM LEFT TO RIGHT ON THE SOURCE LINE) ARE THE FOLLOWING:

(1) LOCATION FIELD

(2) INSTRUCTION FIELD

(3) OPERAND FIELD

(4) COMMENT FIELD

THE LOCATION FIELD CONTAINS A MNEMONIC WHICH IS ASSIGNED

THE VALUE OF THE ADDRESS OF THE LOCATION WHICH THE SOURCE LINE REPRESENTS (UNLESS THE INSTRUCTION FIELD CONTAINS ONE OF THE PSEUDO-OPS $EQU, $OPD, OR $OPDM). IF THE FIRST CHARACTER ON THE LINE IS A SPACE, THE LOCATION FIELD IS NOT PRESENT.

THE INSTRUCTION FIELD CONTAINS ONE OF THE FOLLOWING:

(1) A PSEUDO-OP SYMBOL,

(2) A MNEMONIC WHICH REPRESENTS AN INSTRUCTION WHICH REQUIRES AN OPERAND, OR

(3) AN OPERANDLESS INSTRUCTION MNEMONIC OR A SET OF THESE MNEMONICS SEPARATED BY PLUS SIGNS (+), WHICH DENOTE "INCLUSIVE OR" IN THIS FIELD.

IF THE INSTRUCTION FIELD CONTAINS AN OPERANDLESS INSTRUCTION THE OPERAND FIELD IS NOT PRESENT. INDIRECT ADDRESSING IS INDICATED BY AN ASTERISK (*) APPENDED TO THE RIGHT OF A MNEMONIC WHICH REPRESENTS AN INSTRUCTION WHICH REQUIRES AN OPERAND.

THE OPERAND FIELD CONTAINS A SET OF PROGRAM SYMBOLS AND/OR NUMBERS SEPARATED BY THE BINARY OPERATOR SYMBOLS "+" (2'S COMPLEMENT ADDITION) AND/OR "-" (2'S COMPLEMENT SUBTRACTION). IN ADDITION, THE FIRST PROGRAM SYMBOL OR NUMBER MAY BE PRECEEDED BY EITHER OF THE UNARY OPERATORS "+" (UNARY PLUS) OR "-" (2'S COMPLEMENT). LITERALS ARE DENOTED BY AN EQUAL SIGN (=) APPENDED TO THE LEFT END OF THE OPERAND FIELD. AN ASTERISK (*) REPRESENTS A MNEMONIC WHOSE VALUE IS THE ADDRESS OF THE LOCATION WHICH THE SOURCE LINE IN WHICH IT APPEARS REPRESENTS (IN THE OPERAND FIELD ONLY). THE LOW ORDER 13 BITS OF THE VALUE OF THE EXPRESSION IN THE OPERAND FIELD ARE ADDED TO THE VALUE REPRESENTED BY THE INSTRUCTION FIELD.

PSEUDO-OP SYMBOLS ARE WRITTEN IN THE INSTRUCTION FIELD AND CONSIST OF A DOLLAR SIGN ($) APPENDED TO THE LEFT OF THE PSEUDO-OP MNEMONIC. THE FOLLOWING SYMBOLS ARE ACCEPTED BY THE ASSEMBLER:

$DC A WORD WHICH CONTAINS THE FULL 18-BIT VALUE OF THE EXPRESSION IN THE OPERAND FIELD IS PRODUCED.

$DS THE 18-BIT VALUE OF THE EXPRESSION IN THE OPERAND FIELD IS ADDED INTO THE LOCATION COUNTER WITHIN THE ASSEMBLER (BY TWO'S COMPLEMENT ADDITION). (ALL MNEMONICS IN THE OPERAND FIELD MUST BE PREDEFINED.)

$END THE END OF THE SOURCE PROGRAM IS DECLARED.

$EQU THE PROGRAM SYMBOL MNEMONIC IN THE LOCATION FIELD IS ASSIGNED THE 18-BIT VALUE OF THE EXPRESSION IN THE OPERAND FIELD. (ALL MNEMONICS IN THE OPERAND FIELD MUST BE PREDEFINED.)

$OPD THE OPERANDLESS INSTRUCTION MNEMONIC IN THE LOCATION FIELD IS ASSIGNED THE 18-BIT VALUE OF THE EXPRESSION IN THE OPERAND FIELD. (ALL MNEMONICS IN THE OPERAND FIELD MUST BE PREDEFINED.)

$OPDM THE OPERAND-REQUIRING INSTRUCTION MNEMONIC IN THE LOCATION FIELD IS ASSIGNED THE 18-BIT VALUE OF THE EXPRESSION IN THE OPERAND FIELD. (ALL MNEMONICS IN THE OPERAND FIELD MUST BE PREDEFINED.)

$ORG THE LOCATION COUNTER WITHIN THE ASSEMBLER IS SET TO THE 18-BIT VALUE OF THE EXPRESSION IN THE OPERAND FIELD. (ALL MNEMONICS IN THE OPERAND FIELD MUST BE PREDEFINED.)

$TEXT THE FIRST CHARACTER IN THE OPERAND FIELD IS TAKEN AS A BREAK CHARACTER, AND ALL CHARACTERS TO THE RIGHT OF IT UP TO THE NEXT BREAK CHARACTER ARE PACKED AS 3 6-BIT CHARACTER CODES PER WORD. IF THE NUMBER OF CHARACTERS BETWEEN THE BREAK CHARACTERS IS NOT A MULTIPLE OF 3, THE LAST WORD GENERATED IS PADDED WITH NULL CHARACTER CODES (77).

$TITLE ALL CHARACTERS TO THE RIGHT OF THIS PSEUDO-OP ARE TAKEN TO BE THE TITLE OF THE CURRENT SECTION OF THE PROGRAM. (THIS TITLE IS TYPED ON THE TELETYPE DURING PASS 1 OF THE

ASSEMBLY, BEGINNING WITH THE FIRST NON-BLANK CHARACTER.)

THE ASSEMBLER IGNORES SOURCE LINES WHICH BEGIN WITH AN ASTERISK (*), SOURCE LINES WHICH HAVE NO FIELDS, AND COMMENT FIELDS.